E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Design and prototyping of an automated system for the fabrication and testing of optical fiber sensors

Grado en Ingeniería
en Tecnologías Industriales

Trabajo Fin de Grado

Autor: Lucía Teruel Saldaña

Tutor: Javier Goicoechea Fernández y Adrián Vicente Gómara

Pamplona, 27 de agosto de 2019

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

ABSTRACT

This thesis describes the design and programming of the movement and temperature control of an automated system used for the fabrication of optical fiber sensors. The automated system pumps and injects different programmed samples into a flow cell containing the optical fiber, which makes the sensor. The different samples are displayed on a rectangular matrix in a Cartesian coordinate robot. The temperature of the system is measured by a thermistor, adjusted with a couple of Peltier cells and controlled with a L298N motor driver.

The system is controlled by an Arduino Mega and it also includes a printed Arduino Mega shield with the designed circuits for the movement and temperature control. The user is able to move and program the system with a Raspberry touchscreen using CNC commands.

The design and coding of the system enable the use of different sensors, motors and other components; and also, the use of different samples for the optical fiber sensor.

KEYWORDS

Automated system, biosensor, stepper motor, motor driver, PI control, CNC code

INDEX

FIGURE INDEX

1. INTRODUCTION

This project has been developed to simplify and optimize the production of biosensors.

When working with biological components, the preparation and handle of the solutions, and experimental procedures are carefully done. Manipulation of biological substances is, most times, performed manually and, therefore, the likelihood of human error increases, which can damage the final product or the final results of the experiment. Automated systems reduce this potential risk, increase productivity and make a more efficient use of the materials/components used. The use of these systems also makes the process more stable, with greater control on the final product's quality.

The system is the following:

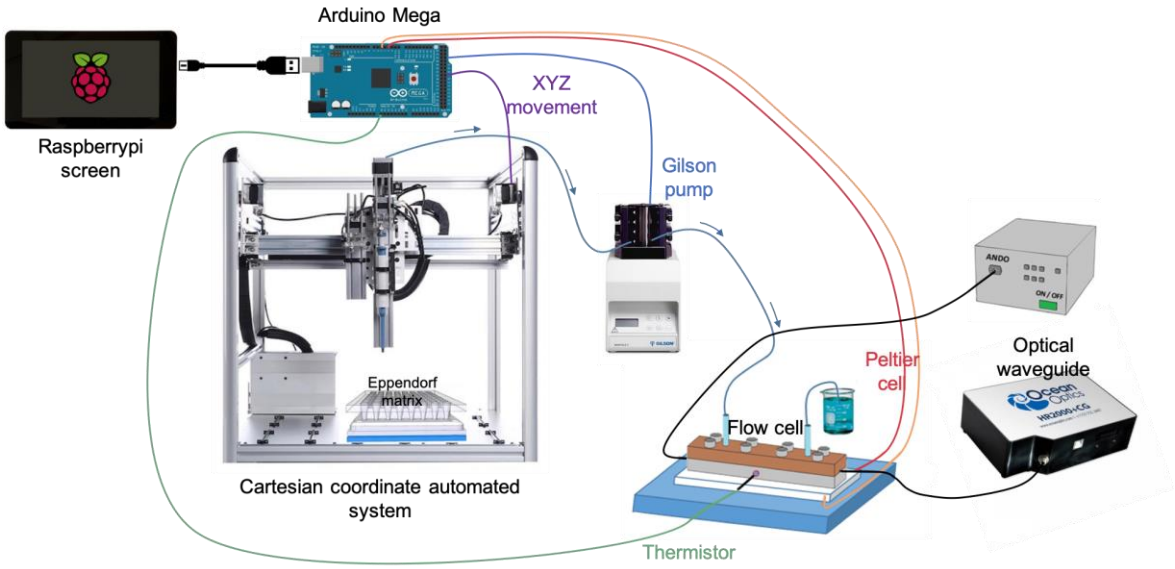


FIGURE 1 - OVERALL SYSTEM

2. OBJECTIVES

The main objective of this project is to design and elaborate a Cartesian automated system for the fabrication of biosensors with optic fiber, based on a previous Master's Thesis developed by the engineer Adrián Vicente. The system must be robust, easy to handle and flexible; this will allow the system to work with different kinds of biosensors. By automatizing the process, the optical fiber sensors would increase their quality and the process will become more efficient.

The two main goals of this project are:

- Design and implement a system that controls the XYZ-movement of the Cartesian coordinate automated system.
- Design and implement a system that controls the temperature of the Flow Cell.

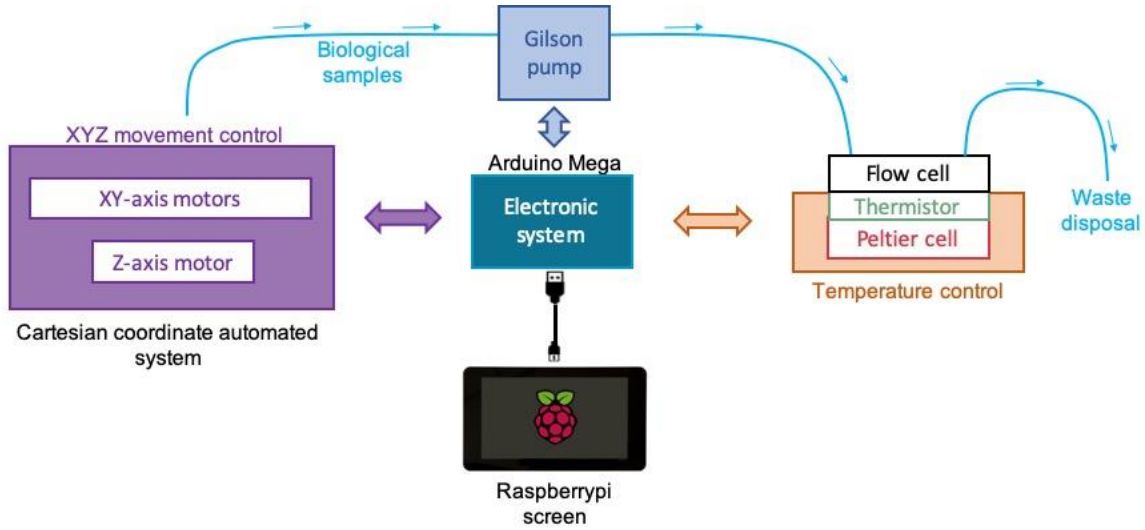


FIGURE 2 – SYSTEM'S BLOCK-DIAGRAM

The achievement of these two goals covers the study and experimental procedures for the development of the hardware and software of both systems. Also, the programming and electronic design of the two parts. And finally, the integration of the two systems into one final system.

3. HISTORICAL BACKGROUND

Since the electrification of factories at the beginning of the 20th century, the mass production of goods has been a reality. This allowed products to be manufactured on a large scale in less time and with less money. Henry Ford not only popularized this type of manufacturing, but he also included several electromechanical systems on its lines to speed up the process and ease the labor. For that reason, the term "automation" is attributed to one of its company's engineers [1].

Automation is the use of automatic equipment in a manufacturing process that replaces or reduces the decision-making of the operator and requires less or no human assistance.

The first mechanical systems were made with pulleys, weight and levers. But these systems still required human strength to function. The next step was the creation of machines powered by running water, air or steam, which date back to more than 2,000 years ago. Starting in the late 18th century, the first automated machines for the cloth industry were created; this made the production faster and cheaper [2]. Later, the first programmable machine was introduced by Joseph-Marie Jacquard. With the use of steel punch cards, he was able to generate complex patterns in textiles.

The first steam-powered engine led to the first industrial revolution during the first half of the 19th century; a brand-new era for the manufacturing of goods. Later that century the concept of a machine capable of analyzing and processing data was proposed by Charles Babbage [1].

Numerical control (CN) or Computer Numerical Control (CNC) was developed during the 1940s and 1950s and was first coded using punched tape. It was used to control tools moved by motors, following coded points specified in x-y-z Cartesian coordinates. John Parsons was the first to come up with this control when he was researching new ways of improving the manufacturing of helicopter blades. This led to research projects held at the Massachusetts Institute of Technology (MIT), were the first numerical control machine was exhibited [3].

When integrated circuits were developed, in the 1960s, machines became more efficient, smaller and cheaper with time [1]. Also, new developments in data storage and sensor technology, and the development of the theory of control systems, made the control of automated machines easier, adaptable and more precise.

Not only the industry of manufacturing goods changed, but the laboratory industry did as well. Prior to the 1950s most laboratory automated systems were hand made by users. When commercial electronic components became available the industry changed. The use of these components with sensors, favored, during the second half of the 1950s, the first systems that could perform analytical analysis on loaded samples or collected data. In 1984, Daniel J. Macero and Brian J. McGrattan, analytical chemistry professor and graduate student (respectively) from Syracuse University, developed the first laboratory programmable robotic arm capable of picking up, placing and moving lab equipment [4].

Since then, numerous new technologies have been developed, but it was not until recently that those technologies have been made available to everyone.

This can also be seen in 3D printers. In 1981 Hideo Kodama designed a new rapid-prototyping machine that could print in layers. Three years later, Charles Hull invented the stereolithography, and some years later the first SLA (stereolithography apparatus) and SLS (Selective laser sintering) machines were created [5].

In 2005 Adrian Bowyer created the RepRap project, which changed the 3D printing world. That meant that basically anyone could buy, build or even print their own 3D printer; and with it, basically build anything.

Nowadays anyone can build their own CNC machine, pick and place system or 3D printer. The low cost of components, the use of the internet and the appearance of universal open-source electronic modules, code and platforms, have made this type of technology available to everyone. The use of these technologies brings many advantages such as productivity, better quality, improved safety, more efficient use of materials and avoids human error.

4. HARDWARE ELEMENTS

As explained before, see Figure 2, the two main goals of the thesis are the design and implementation of two systems:

- XYZ–movement control
- Temperature control

The following elements are the key hardware ones used for each system.

4.1. XYZ-MOVEMENT CONTROL SYSTEM

4.1.1. STEPPER MOTORS

For this project, these motors are going to be used for the XY-axis movement.

Stepper motors are electric DC motors that rotate in very accurate discrete increments or "steps". These are used in applications where high torque and very precise positions are needed. For example, in 3D printers, stepper motors are used to precisely position the printhead [6]. *Other examples are: CNC machines, DVD and Blu-ray players, and camera zoom lenses.*

TYPES AND CHARACTERISTICS

Internally, these motors are composed of a magnetized geared shaft surrounded by coils that act as electromagnets. There are two types of stepper motors depending on their coil wiring arrangement: bipolar or unipolar. For this project bipolar stepper motors have been chosen as they have reverse voltage polarity, which allows to reverse direction, and also a higher torque.

STEPS

The smallest motor rotation a stepper motor can achieve is called a "step", and the shaft movement performed in one step is measured in degrees (step-angle). Eq. 1 can be used to obtain the nº of steps the motor can perform in one revolution:

$$\frac{360^o\ (1\ revolution)}{step\ angle\ (degrees\ per\ step)} = n^o steps\ per\ revolution$$

EQ. 1

TORQUE

When choosing a motor, the amount of force provided by it is also very important. Two aspects should be taken into account when taking about torque:

- Holding torque: The amount of force the stepper is able to supply when energized.
- Detent torque: Amount of torque the motor has when the stepper is NOT energized.

MOTOR CONTROL

Each step or microstep is achieved by applying a pulse, supplied by the driver, to the internal coils of the motor. After each step, the motor holds its position.

SPEED

The speed (rpm) of the motor is proportional to the pulse's frequency [7]: higher frequency means more speed and a lower frequency means lower speed.

ROTATION

The rotation (degrees) that the stepper performs is controlled with the number of pulses provided to it. One pulse means one step. In our case, the chosen stepper motors had a basic step angle of 1.8º/step.

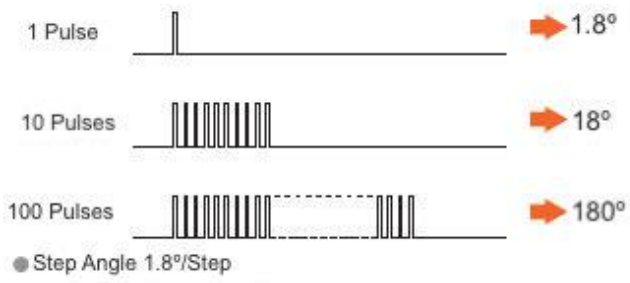


FIGURE 3 - ROTATED ANGLE PER PULSE. RETRIEVED AND MODIFIED FROM: HTTPS://WWW.ORIENTALMOTOR.COM/STEPPER-MOTORS/INDEX.HTML#STEPPERMOTORS

POSITION REFERENCE OR HOMING

Steppers can move to a certain position, taking as reference their starting position. These motors are "unaware" of their position; this means that they can be misaligned if an external force is applied on them. That is why most steppers are first driven to a "homing" or reference position.

4.1.2. POLOLU STEPPER DRIVER BOARD (DRV8825)

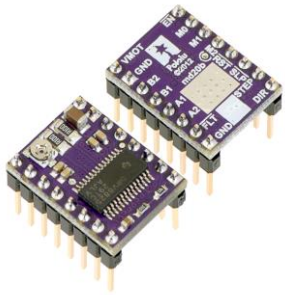


FIGURE 4 - DRV8825 STEPPER MOTOR DRIVER. RETRIEVED FROM: HTTPS://WWW.POLOLU.COM/PRODUCT/2133

This driver allows the control of a bipolar stepper motor with an output current up to 2.2 A per coil [8].

CHARACTERISTICS

Some of these drivers' key features are the following ones:

- Adjustable current control (with the help of a built-in potentiometer)
- Automatic chopping control (automatic current decay mode, fast or slow)
- Over-temperature, over-current and under-voltage protection.
- 45 V maximum supply voltage (12 V for this project).

LIMITATIONS

The voltage supplied by the driver to the stepper motor might be higher than stated, but supplied current must **never** be higher than stipulated, otherwise it will not work properly and there will also be risk of damaging the stepper [9].

The supplied current can be easily limited using the built-in potentiometer on the driver. This process was done according to the stepper specifications and it is later detailed on the experimental procedure (5.1.2 Connections and limit current, p.9).

DRIVER-STEPPER CONNECTION

The following overview of the basic connections between the driver and the stepper can be found on the driver's webpage, see [9]:

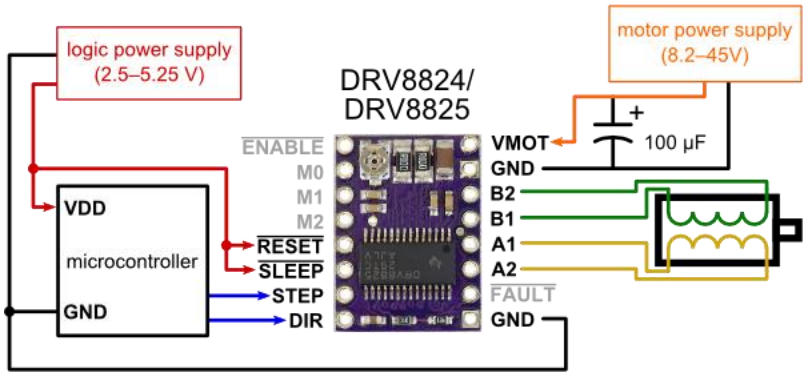


FIGURE 5 – BASIC DRV8825 CONNECTIONS. RETRIEVED FROM: HTTPS://WWW.POLOLU.COM/PRODUCT/2133

Figure 5 was used in the driver's connection to the motor. This part is later explained in detail.

4.1.3. SERVO MOTOR

For this project, this motor is going to be used for the Z-axis movement.

Servo motors are DC motors designed to move a given angle and hold this position. Its range of motion goes from 0 º to 180 º, some of them even up to 360 º [10].

Some of the key features why this type of motors is so widely used in robotics are the following:

- Rotate a specific number of degrees
- DC current motor
- Built-in control circuit
- Precise position control

The wiring of these motors is considerably easy as it only has 3 connections:

TABLE I
SERVO CONNECTIONS

Wire color	Connection
Red	5V – power supply
Black/brown	GND – ground
Yellow/white	PWM Control signal

The control of the motor is done by a PWM signal, in which pulse width indicates the desired angle of the motor's shaft.

4.2. TEMPERATURE CONTROL SYSTEM

4.2.1. THERMISTOR

Thermistors are temperature sensors that vary their internal resistance with temperature. Three of the reasons why these sensors are so widely used are that they are predictable, cheap and easy to use [11]. There are two types of thermistors depending on how their internal resistance behaves with temperature:

- Negative Temperature Coefficient (NTC) -> resistance inversely proportional to temperature
- Positive Temperature Coefficient (PTC) -> resistance proportional to temperature

For this project, the type of thermistor used is an NTC one.

4.2.2. PELTIER CELL

A Peltier cell is a thermoelectric device that is able to generate a temperature difference between the two faces (one side hot and the other one cold) of a semiconductor when there is current flowing through it [12]. Semiconductors can achieve this effect because they can conduct electricity, but they are poor heat conductors. This effect is reversible, and that can easily be achieved just by inverting the polarity of the current.

In order for the Peltier cells to work properly, a heat dissipation system must be implemented. This system allows the heat to exit the hot side without damaging the cell.

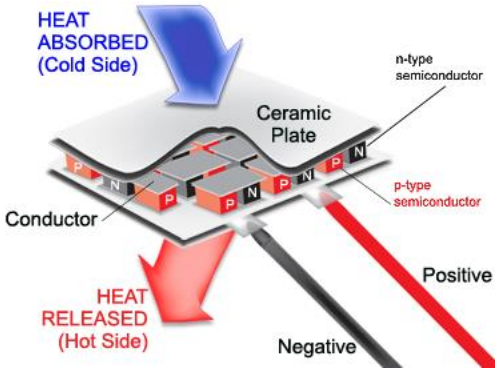


FIGURE 6 - PELTIER CELL HEAT TRANSFER. RETRIEVED FROM: HTTPS://WWW.SILRAM.CO.IL/PRODUCT/TEC/

4.2.3. L298N MOTOR DRIVER

The L298N H-bridge motor driver enables the speed and direction of rotation control of a DC motor. For this project, a L298N motor controller board is going to be connected to a Peltier cell. The two H-bridge that compose the system allow the control of the direction of the current supplied to the Peltier cell and the speed of heating/cooling of the system. Each H-bridge is composed of 4transistors that enable to reverse the direction of the current [13], and so, the working mode of the Peltier cell (cooling/heating).

The range of work of the module is between 3 and 35 V, and with a supplied current of up to 2 A. This a widely used device due to its simple control and reduced price.

5. Experimental procedure – XY Axis movement

The first part of the project consists of the control of the XY axis movement.

5.1. Motor first setup

The first step was to study and learn how to control the stepper motor with the chosen driver.

5.1.1. Hardware

- Arduino RobotDyn Uno - CH340/ATmega328PA
- DRV8825 driver
- Nema17 Stepper motor
- 100µF capacitor
- Multiple wires
- Promax power supply (for 12V)
- Multimeter
- Ariston board

5.1.2. Connections and limit current

First, with the help of the Ariston board, and taking Figure 5 as a reference, the basic connections between the driver, the Arduino and the Promax power supply were made. Nothing was turned on yet. Then the limit current for the stepper was adjusted, following the procedure described next:

As explained before, the stepper has a current limit that should not be surpassed because it could damage it. For this experimental procedure, the stepper used had a limit current of 1.8 A. With the help of Eq. 2, obtained from the driver's webpage [9], the current flowing through the stepper can be known by just measuring the voltage on the driver's potentiometer.

$$I_{stpr} = 2 * V_{pot}$$

Eq. 2

As it is mentioned above, the stepper limit current is 1.8 A, so the voltage on the potentiometer must be 0.9 V or lower. As seen in Figure 7, 0.642 A is the closest value achieved, as the potentiometer was very sensitive.

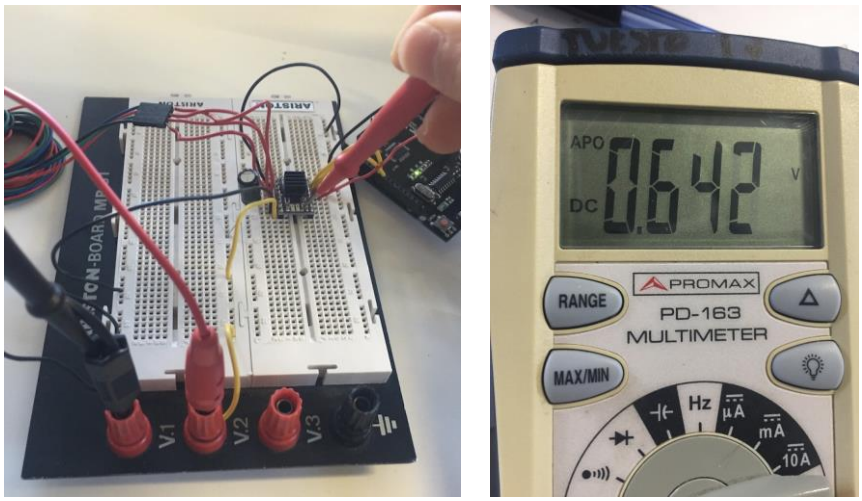


Figure 7 - Manual check of the potentiometer's voltage

Next, the motor was wired to the driver following the next figure, see [14]:

TABLE II
STEPPER WIRING

A+	A-	B+	B-
Black	Green	Red	Blue

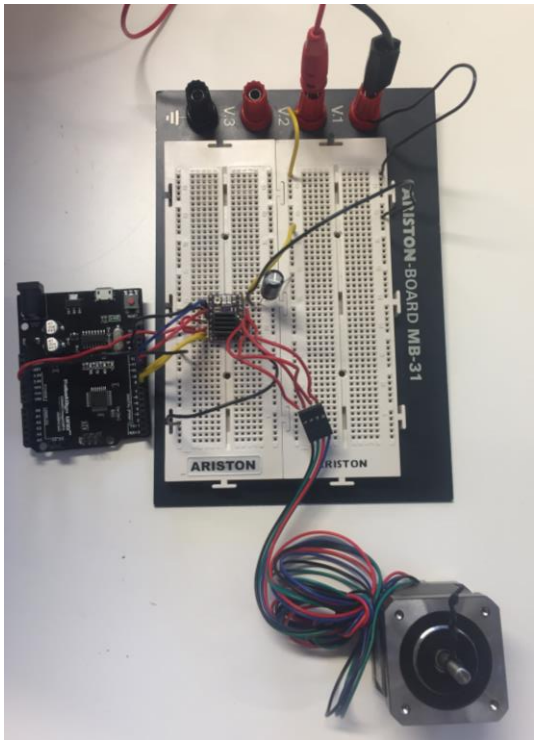


FIGURE 8 - REAL BASIC DRIVER CONNECTIONS MADE

Finally, a simple Arduino test code was transferred to the Arduino board and the motor power supply was turned on.

5.2. ARDUINO TIMER INTERRUPTS TESTING

By using interrupts, we allow the controller to run more smoothly. When an interrupt occurs, the controller stops the action it was doing, saves its actual state, performs the code related to the interrupt, reloads the saved state and continues where it was left at.

2 types of interrupts:

- Internal: these occur inside the microcontroller (Ex: timers)
- External: these are triggered outside the microcontroller (Ex: buttons)

The Arduino microcontroller has 3 different timers:

- Timer 0: 8bits [0-255]
- Timer 1: 16bits [0-65535]
- Timer 2: 8bits [0-255]

TABLE III
TYPE OF TIMERS

NAME	SIZE	POSSIBLE INTERRUPTS	USES IN ARDUINO
TIMER0	8 bits (1byte)	- Compare match - Overflow	- delay(); milis(); micros(); - analoWrite() pins 5, 6
TIMER1	16 bits (2bytes)	- Compare match - Overflow - Input capture	- Servo functions - analoWrite() pins 9, 10
TIMER2	8 bits (1byte)	- Compare match - Overflow	- tone() - analoWrite() pins 3, 11

Table retrieved from: https://www.youtube.com/watch?v=J61_PKyWjxU&t=657s [15]

Each timer increases its count by 1 each time its oscillator strikes, by adjusting the pre-scaler the timing can be made to last longer:

$$TIMERcount + 1 = \frac{prescaler\ [1, 8, 64, 256, 1024]}{frequency\ of\ the\ oscillator}$$

EQ. 3

Later on, the code used to control de steppers will use internal-timer interrupts, so in order to get more familiar with that concept the following code was developed:

TIMER1 interruption – Velocity of the motor - 2kHz wave – pin9

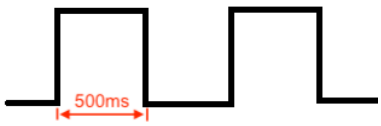


FIGURE 9 – PIN9 OUTPUT

In order to control the velocity of the motor at 2 kHz, the TIMER1 interruption was set to trigger every 500ms. The timer was set on comparison mode with a pre-scaler of 1 and a value to be compared of 7999.

$$TIMER1 comparison\ value = \frac{\frac{1}{2x10^3}}{\frac{1}{16x10^6}} - 1 = 8000 - 1 = 7999$$

EQ. 4

The Arduino registers were modified as follows:

```
TCCR1A = 0; // set entire TCCR1A register to 0
TCCR1B = 0; // same for TCCR1B
TCNT1 = 0; // initialize counter value to 0
OCR1A = 7999; // set comparison value (must be <65536)
TCCR1B |= (1 << WGM12); // turn on CTC mode (Comparison mode)
TCCR1B |= (1 << CS10); // set CS10 bit for 1 pre-scaler
TIMSK1 |= (1 << OCIE1A); // enable timer compare interrupt
```

The function related to the TIMER1 interrupt was the following:

```
ISR(TIMER1_COMPA_vect){
//TIMER1 interrupt - VELOCITY control (pin9)
//generates pulse wave of frequency 2KHz
 if (toggle1){
  digitalWrite(9,HIGH); //500ms up
  toggle1 = 0;
 }else{
  digitalWrite(9,LOW); //500ms down
  toggle1 = 1;
 }
}
```

TIMER2 interruption – Direction of the motor – 1/3Hz wave – pin8

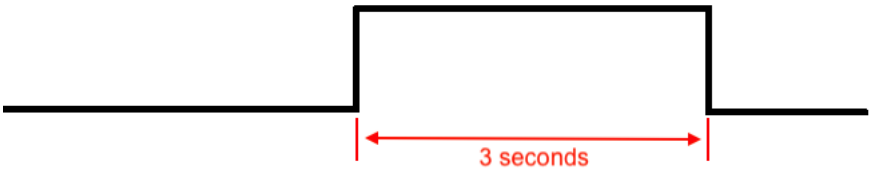


FIGURE 10 – PIN8 OUTPUT GOAL

In order to change the direction of the motor every 3 seconds, TIMER2 interruption was set to trigger every 11.9 ms and then introduced into an "if" loop to make it last 3 s. The timer was set on comparison mode with a pre-scaler of 1024 and a value to be compared of 185.

$$TIMER2 comparison\ value = \frac{0.0119}{\frac{1024}{16x10^6}} - 1 = 186 - 1 = 185$$

EQ. 5

The Arduino registers were modified as follows:

```
TCCR2A = 0; // set entire TCCR2A register to 0

TCCR2B = 0; // same for TCCR2B

TCNT2 = 0; // initialize counter value to 0

OCR2A = 185; // set comparison value (must be <256)

TCCR2A |= (1 << WGM21); // turn on CTC mode (Comparison mode)

TCCR2B |= (1 << CS22); // Set bits for 1024 pre-scaler

TCCR2B |= (1 << CS21);

TCCR2B |= (1 << CS20);

TIMSK2 |= (1 << OCIE2A); // enable timer compare interrupt
```

The function related to the TIMER2 interrupt was the following:

```
ISR(TIMER2_COMPA_vect){
//TIMER2 interrupt - DIRECTION control wave (pin8)
//generates pulse wave of frequency 15.6kHz*5 = 4kHz
 aux=aux+1;
 if(aux==252){ //When we reach 3s (252*0.012s)
  if (toggle2){
   digitalWrite(8,HIGH); //3s up
   toggle2 = 0;
  }else{
   digitalWrite(8,LOW); //3s down
   toggle2 = 1;
  }
  aux=0;
 }
}
```

5.3. CNC CODE TESTING

The goal of the final code was for it to be able to read CNC commands from the serial monitor and properly execute them.

The code used in the final project is a modification of two uploaded codes to GitHub (both codes read CNC commands):

- GcodeCNCDemo6AxisRumbaTimerInterrupt by Dan Royer (i-make-robots) [16]
- GCodeAFMotorV2 by Oliver Beck (hirnwunde) [17]

Prior to their modification, both codes were tested and understood (to be able to merge them).

5.3.1. GCODECNCDEMO6AXISTIMERINTERRUPT

This code is very interesting because it uses timer interrupts to control the motors, just as we had previously tested.

Some very interesting key features of this code are the following:

- Can control up to 6 axes/motors
- Compatible with DRV8825 driver
- Timer interrupt stepper control
- Reads CNC commands
- Adjustable motor speed
- 2 control modes: Absolute and relative

The absolute and relative modes mentioned above didn't work as

ABSOLUTE MODE

Takes as reference-zero-position the stating position, always. Taking this position as a reference then moves the stepper the input number of steps with command "G00 or G01".

Absolute mode can be set using "G90" command

RELATIVE MODE

Takes as reference-zero-position the one set with command "G92". Taking this position as a reference then moves the stepper the input number of steps with command "G00 or G01".

Relative mode can be set using "G91" command.

5.3.2. GCODEAFMOTORV2

This code is very interesting because it has a homing function that moves the steppers to its home position. It is based on the previous one, but it also has some new features:

- Has "homing" function
- Has 2 limit switches & 2 homing switches
- No timer interrupt motor control

When testing this code, we realized that the Absolute and Relative modes worked as in the previous one.

5.4. CNC FINAL CODE

As it is said before, the code used in the final project is a modification of two uploaded codes to GitHub:

- GcodeCNCDemo6AxisRumbaTimerInterrupt by Dan Royer (i-make-robots) [16]
- GCodeAFMotorV2 by Oliver Beck (hirnwunde) [17]

Dan Royer's code was taken as the base and parts from Oliver Beck's code were added to it. The overall functioning of the merged code is shown in Figure 11.

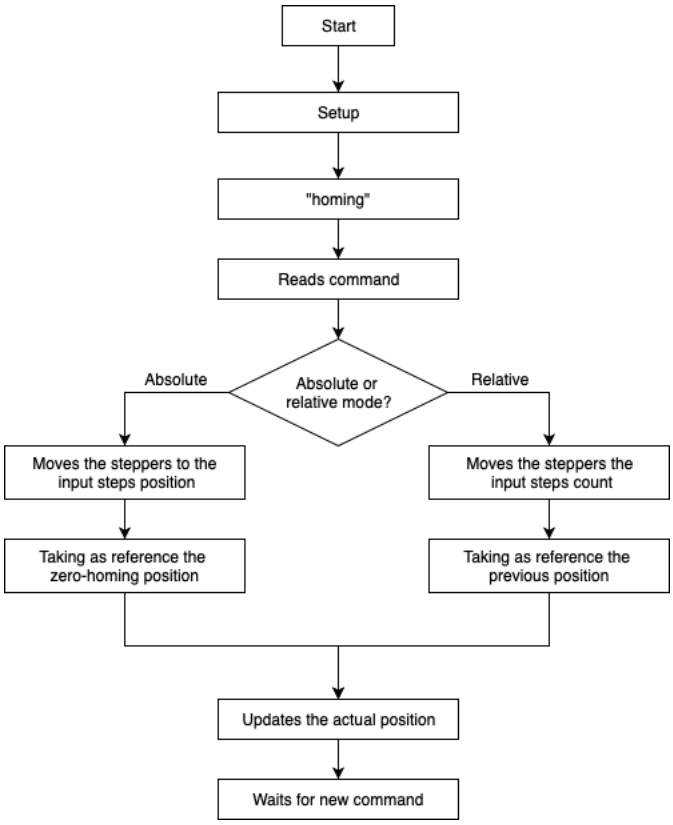


FIGURE 11 - OVERALL FUNCTIONING FLOW CHART

Next, are some of the greatest changes or checks made to the original code:

5.4.1. REMOVAL OF UNUSED FUNCTIONS AND VARIABLES

As the two original codes were thought for 3D-printing purposes, they both had some functions that were not going to be used for this project, and for so these were removed. For example: "arc()" no need for an arc function.

5.4.2. FUNCTION/PARAMETER ADJUSTMENTS

Some code parameters and functions had to be adjusted in order to work with our setting:

- The onestep() function was modified to take into account the direction of the one-step movement (with the variable "direction").
- The where() function was modified to show the current motor position whenever the stepper made a new movement.
- The G00/G01 command was modified to save the current motor position once it was reached (with function actual_position()).

5.4.3. Functions addition

Some new functions were also added:

- homecycle(): Homing function that moves the stepper to its home position when called. Triggered when command G28 is introduced by the user.
- motor_enable(): Energizes the motors so the user can NOT move them manually. Triggered when command G17 is introduced by the user.
- motor_disable(): Stops energizing the motors so the user CAN move them manually. Triggered when command G18 is introduced by the user.

5.4.4. Max/min feed rate (motor speed) check

The maximum and minimum stepper speeds had to be manually checked in order to set the limits. The following limits were found:

- Maximum stepper speed = 150 steps/second
- Minimum stepper speed = 1 step/second

5.4.5. Relative mode modification

When the code was initially tested, we realized that the relative mode didn't work as expected. The original relative mode worked as an absolute mode, with a modifiable home position (Figure 12), and not as an incremental mode relative to the previous position (Figure 13). In order to solve this problem, the actual position was made to be saved after every movement, and so it could be taken into account for the next move.

The following commands can be used to change between modes:

- G90: Selects the Absolute mode
- G91: Selects the Relative mode

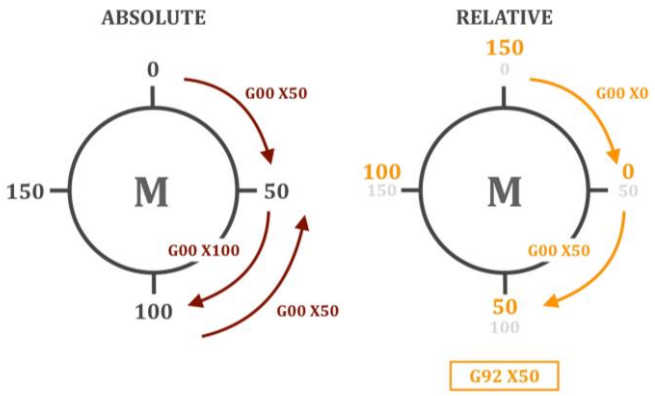


Figure 12 – Original Absolute and relative modes

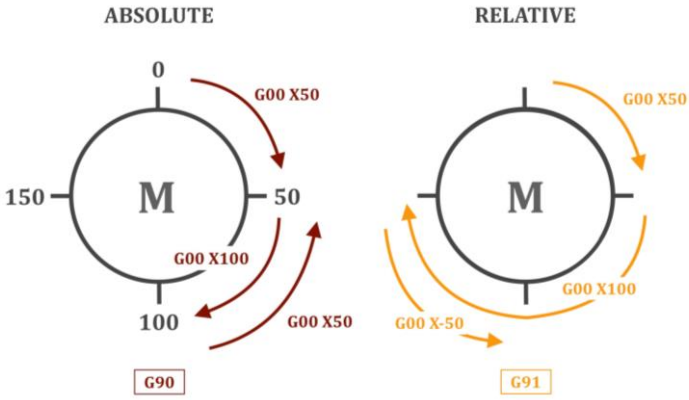


Figure 13 - Final Absolute and relative modes

6. EXPERIMENTAL PROCEDURE – Z-AXIS MOVEMENT

The motor used for the Z-axis is a servo motor, and its implementation is described next.

6.1. HARDWARE AND CONNECTIONS

The connections of the servo are directly made to the Arduino, as this type of motor does not need a driver (see Figure 14).

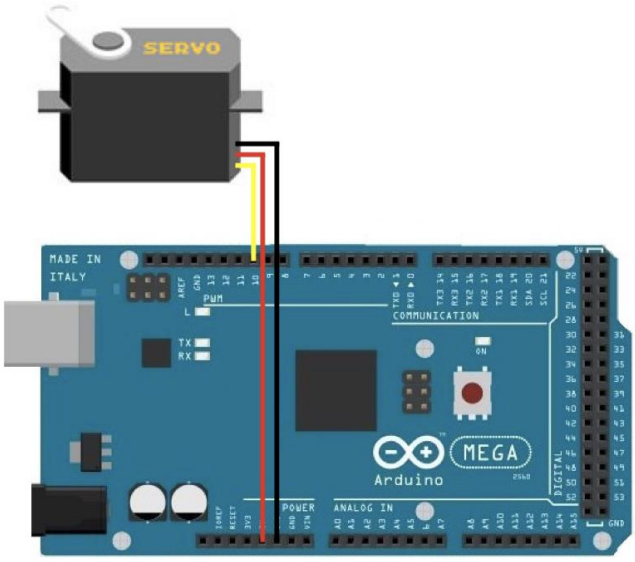


FIGURE 14 - SERVO-ARDUINO CONNECTION

6.2. CODE

The library used for the control of the servo is "**ServoTimer2.h**" [18], which is not the usual library "Servo.h()".

"Servo.h()" works with Timer1 and, for this project, this timer is already in use for the stepper motors for the X and Y-axis. The library "ServoTimer2.h" works with Timer2, and so it allows to use Timer1 for the steppers.

This library can drive up to 8 servos but, for this purpose, there is only one. Usually servos can move between 0-180 º, but when doing some tests with this library it was discovered that, with this code, the steppers can only move between 0-160 º. For this project the servo is not going to move more than 160º, so this is not an inconvenience.

7. EXPERIMENTAL PROCEDURE – TEMPERATURE CONTROL

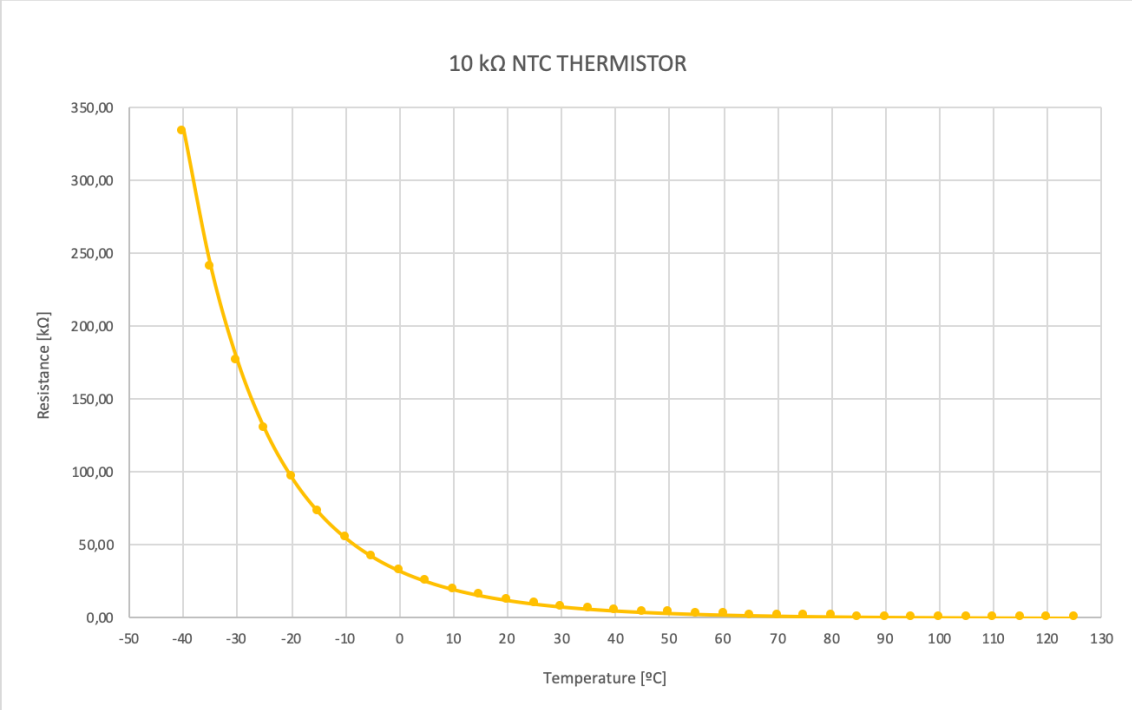
The goal is to control and be able to adjust the temperature of the system with the help of a thermistor, a PI controller and a Peltier cell. As the system is thought to be used for a biological purpose, the estimated temperature working range is thought to be between 15 ºC and 40 ºC. The thermistor circuit was designed with that range of temperatures in mind.

7.1. MATERIALS

The following materials were used for the temperature control module:

THERMISTOR

In order to measure the temperature inside the cell, a thermistor was used. These types of temperature sensors vary their internal resistance with temperature. The thermistor used was an NTC 10 kΩ thermistor NTCLE305E4103SB 10 [19]. NTC thermistors decrease their internal resistance as temperature increases, as seen in Graph 1.



GRAPH 1 - NTC 10KΩ DATASHEET PLOTTED VALUES

PELTIER CELL

The Peltier cell used is an **ET-131-10-13-S Adaptative® Peltier cooler module** [20]**.** This cell has a maximum current input of 3.9 A and a max. voltage of 16.2 V, it can have a maximum temperature difference between both sides of 74 ºC.

PI-CONTROLLER

A Proportional-Integral controller is a control mechanism with a feedback loop that calculates the error between the actual state and the desired state, and calculates a control action upon it. The P term is proportional to the current error of the system and the I term is the integration of past errors over time.

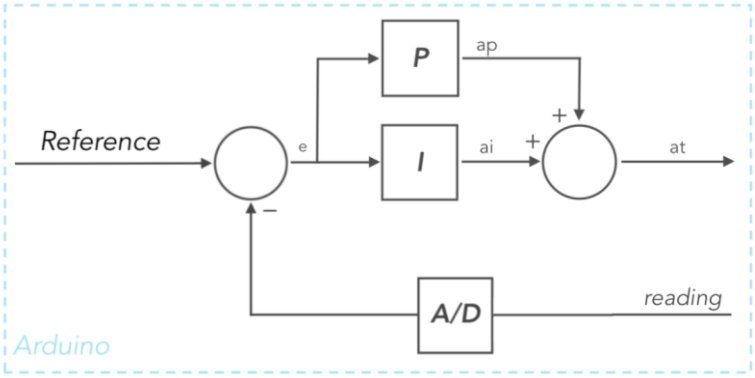


FIGURE 15 - DIGITAL PI CONTROLLER

7.2. THERMISTOR

The selected thermistor takes values from -40 ºC to 125 ºC and the expected working temperatures of our system range between 15 ºC and 40 ºC. The following procedure described next was followed in order to work properly with the thermistor within the system's working range.

7.2.1. MARLIN CODE

Our code is based on a code called "Marlin" [21]. This code is the one used in 3D printers and allows the easy connection of different types of transistors.

The thermistor changes its resistance with temperature, so the value read from it is analog (tension). This tension is then sampled and converted into a digital value. This value is then compared with the reference-table values in the thermistor's table and finally, the temperature is obtained.

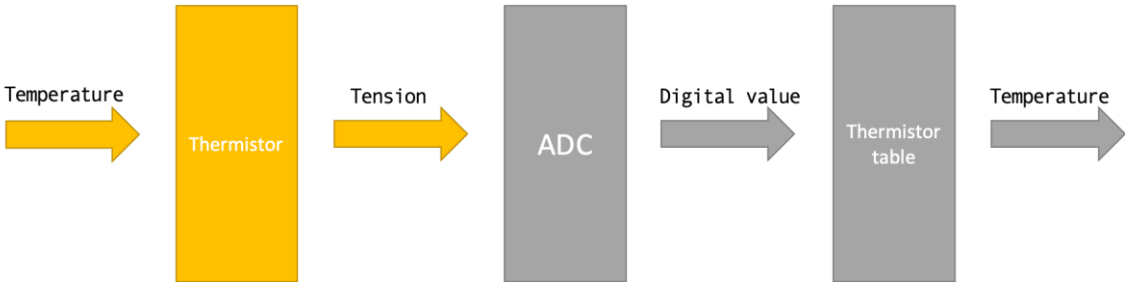


FIGURE 16 - TEMPERATURE ARDUINO PROCESSING

As each temperature sensor is different, each one of them has a different thermistor's table. This table is loaded into the main code and must have 2 columns: the 1st column has the ADC sampled value and the 2nd column has the temperature value related to the ADC value

The procedure followed to obtain this project's thermistor table is later explained in 7.2.3 Table (p.21).

7.2.2. CIRCUIT

The first circuit used to build-in the thermistor was the following:

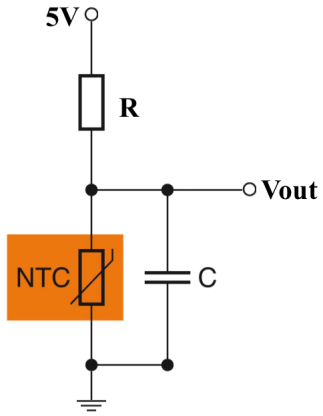


FIGURE 17 - FIRST THERMISTOR'S CIRCUIT

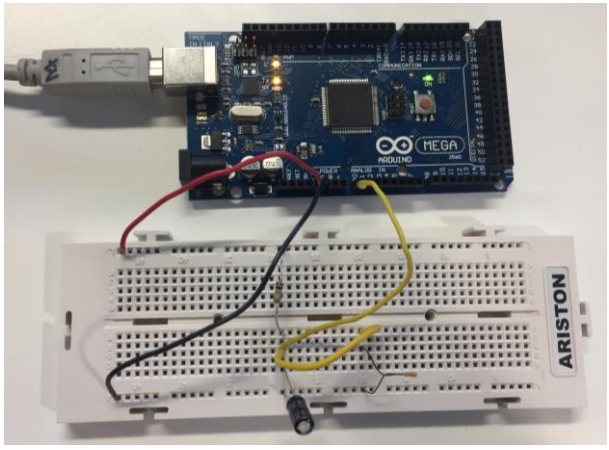


FIGURE 18 - REAL FIRST THERMISTOR'S CONNECTIONS

In order to choose the appropriate R [Ω] and C [F] values, different cases were studied in excel. Finally, the selected components were R = 10 kΩ and C = 10 μF, as these provide the wider range of tension values for temperatures between 15 and 40 ºC.

TABLE IV
VOUT VALUES FOR A 15-40 ºC WORKING RANGE

T [ºC]	$R_{thermistor}$ [Ω]	Vout [V]	ADC 10 bit reading
15	15711,0	3,055	625,12
20	12493,0	2,777	568,19
25	10000,0	2,500	511,50
30	8056,0	2,231	456,43
35	6529,7	1,975	404,11
40	5323,9	1,737	355,42

To improve the sensibility of the system, the Vout values were adjusted to start with near 5 V at 15 ºC and finish close to 0 V at 40 ºC. The final circuit that was implemented is the one in Figure 19 which follows Eq. 6.

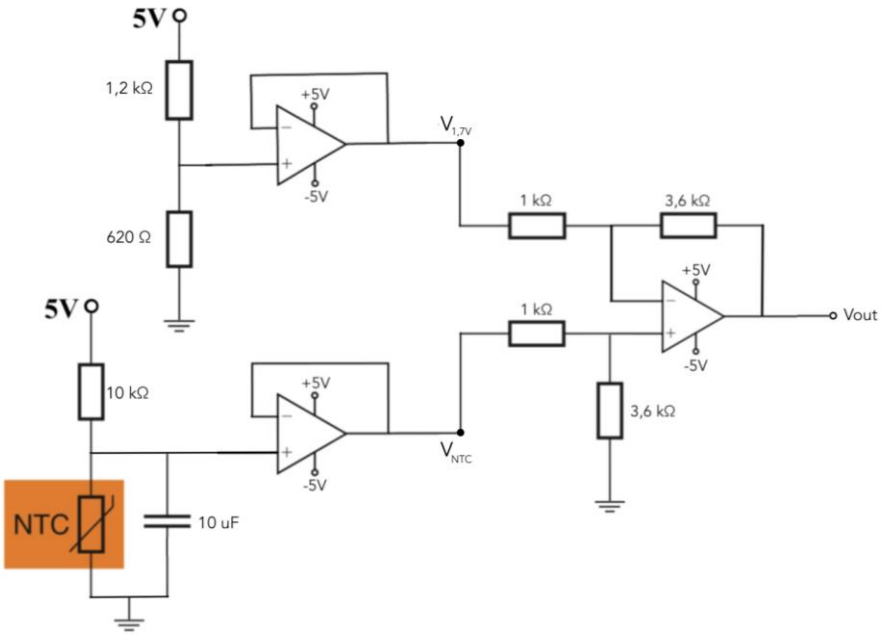


FIGURE 19 - (FINAL THERMISTOR CIRCUIT)

$$V_{out} = G(V_{NTC} - V_{1,7}) = 3{,}6\ x\ 5(\frac{R_{NTC}}{R_{NTC} + 10K\Omega} - \frac{620\Omega}{620\Omega + 1{,}2K\Omega})$$

EQ. 6

As seen in the circuit, 1,7 V is subtracted to the voltage provided by the NTC, then it is multiplied by 3,6. V_{NTC} and $V_{1.7V}$ are followed by voltage followers and all the Operational Amplifiers are supplied by ±5 V.

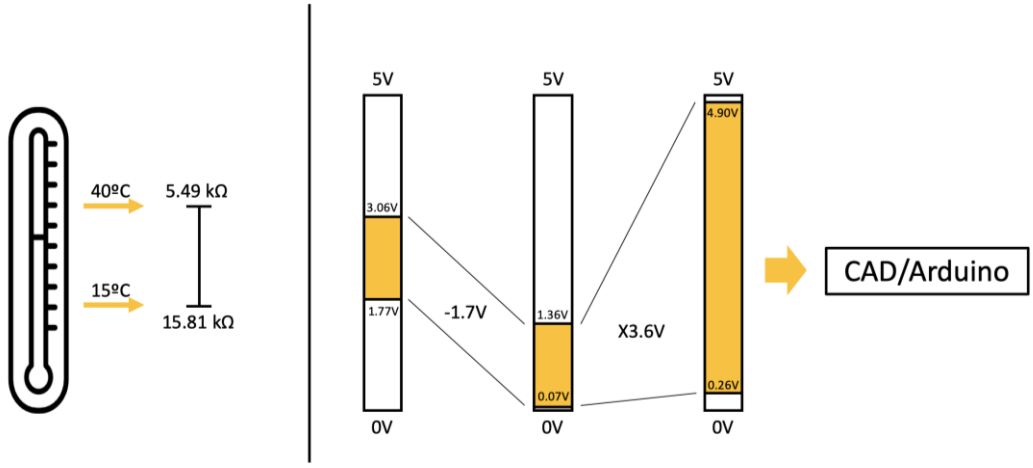


FIGURE 20 - TEMPERATURE CONVERSION

7.2.3. TABLE

As said before each thermistor has its table, which has the corresponding temperature to the read value by the Arduino. The table has two columns, the first one contains the ADC values read by the Arduino and the second column contains the real temperature related to those ADC values multiplied by 10 (see Figure 21).

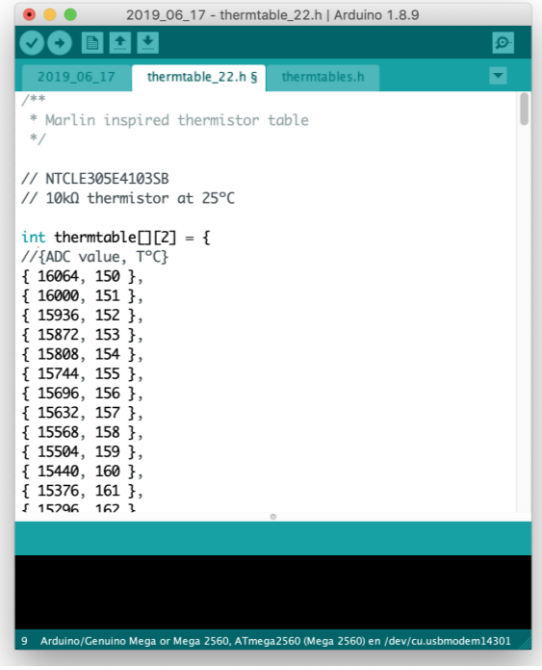


FIGURE 21 - THERMISTOR TABLE FOR THE NTC USED IN OUR CIRCUIT

NOTE: The values found on the table correspond to a 10bit conversion with an over-sampler of 16. Also, by multiplying the temperatures times 10 we can save those values as integers and not floats, which saves us a lot of memory.

The ADC values read by the Arduino must be calculated. This calculation can be done in two ways: using the resistances provided by the thermistor's datasheet or measuring those values experimentally.

In our case, we first tried to use the resistance values from the thermistor's datasheet [19]. The values provided ranged between -40 ºC and 125 ºC and had 5 ºC increments. Our estimated working range is between 15 ºC and 40 ºC, with a 0.1ºC precision, so the values missing from the datasheet were calculated using linear interpolation. Once we tried to measure temperature with those values, we realized that the use of these values led to erroneous temperature measurements of ±4 ºC.

As the datasheet values were not a precise option, we tried the experimental approach. To get the experimental resistance values we used a Peltier cell, which we were able to set precisely to a range of temperatures between 10 and 40 ºC, with the help of a PROMAX voltage source. The thermistor was attached to the Peltier cell using thermal paste and a clip, its resistance was directly measured with a multimeter. The temperature of the Peltier cell was read with the help of a digital thermometer (see the setup in Figure 22).

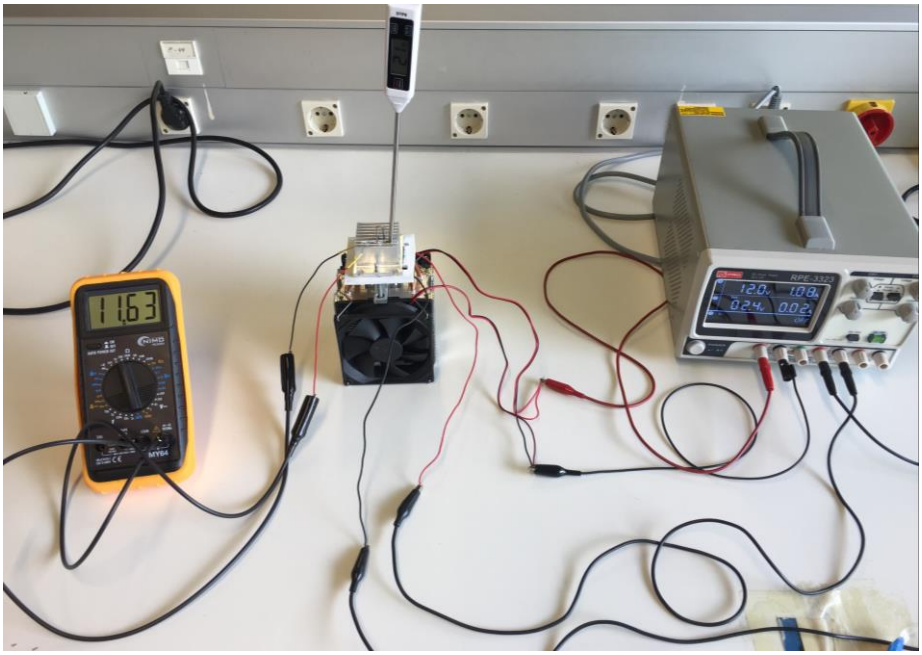


FIGURE 22 - SETTING USED TO MEASURE THE RESISTANCE DIRECTLY

This way we were able to measure precise resistance values every 0.1 ºC. Once these values were obtained, the ADC Arduino values were calculated, and the final thermistor's table was generated. The final setting to measure temperature is the one in Figure 23.

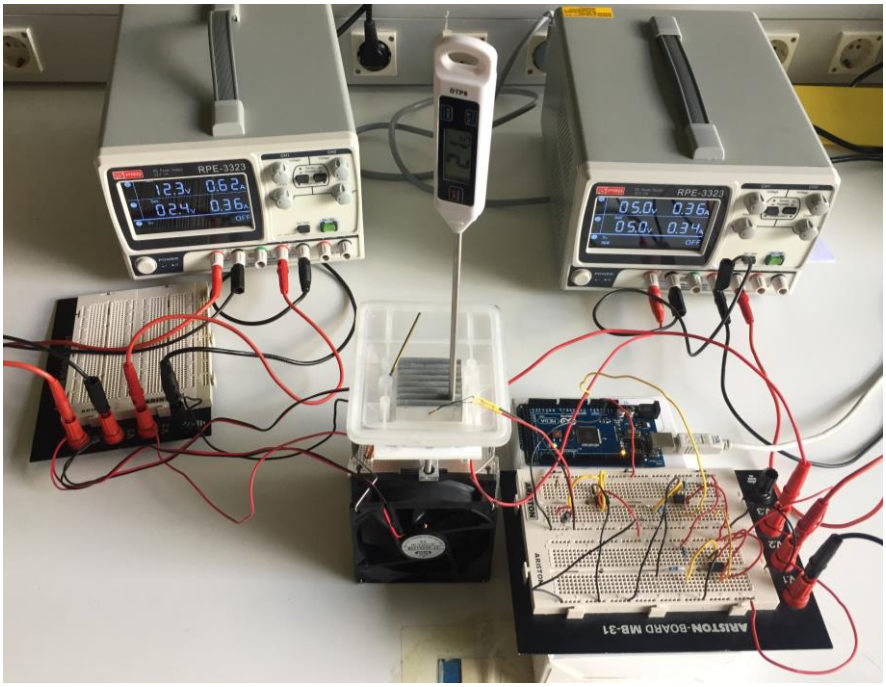


FIGURE 23 - FINAL SETTING OF THE SYSTEM

7.2.4. OUR CODE

Our code was inspired by Marlin code [21] and designed to be able to read thermistor tables saved in separated files outside the main one. Several thermistor-tables can be saved in the main-thermistor code folder, this allows for different thermistors to be used without barely changing the main code.

Also, there is a file called "thermtables.h" which loads the correct thermistor table into the main code, taking into account the thermistor indicated on it.

A table lookup function "table_lookup(adc)" was implemented to search for the closest value in the table to the read one. This function looks for the exact temperature value matching the read one or looks for the closest one. The function enters the table with the read ADC value, first checks if the value belongs to the upper part or lower part of the table (this makes the code to run more smoothly and faster), then looks for an exact match to said ADC value, if not found then looks for the closest one.

NEW THERMISTOR TABLE IMPLEMENTATION PROCEDURE

The procedure to follow when wanting to implement a new thermistor is the following:

1- Create a new thermistor table with a new number (nº) and save it as "thermtable_nº.h".
2- Add the thermistor table name and number to the file "thermtables.h".
3- Change the correct thermistor number (nº) in the main code (#define ThermSensor (nº))
4- Check the rest of the settings to make sure the connection-ports are correctly designated.

7.3. PELTIER CELL

To power the Peltier cell a L298N motor drive was used, to control this driver with the Arduino PWM signals were used.

7.3.1. ARDUINO PWM OUTPUT

The Arduino Mega board has 12 PWM-pins located from pin nº2 to pin nº13. The important characteristic of the PWM is the duty cycle. This is controlled with the following Arduino function:

- analogWrite(pin, DC_value)

The DC_value introduced does not go from 0-100, but from 0-255:

0% Duty cycle = 0 DC_value
100% Duty cycle = 255 DC_value

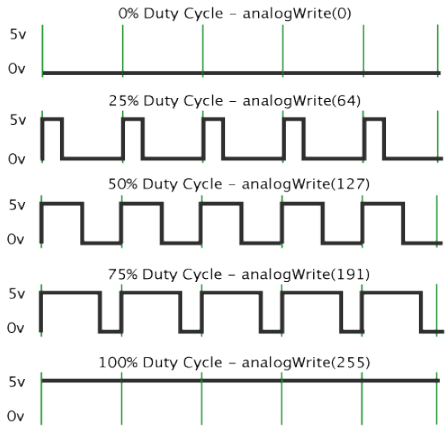


FIGURE 24 - PULSE WIDTH MODULATION WITH ARDUINO. RETRIEVED FROM: HTTPS://WWW.ARDUINO.CC/EN/TUTORIAL/PWM

7.3.2. L298N MOTOR DRIVE

As it is said before, to supply the Peltier cell a L298N motor drive was used, which allows supplying the cell with the proper current and voltage.

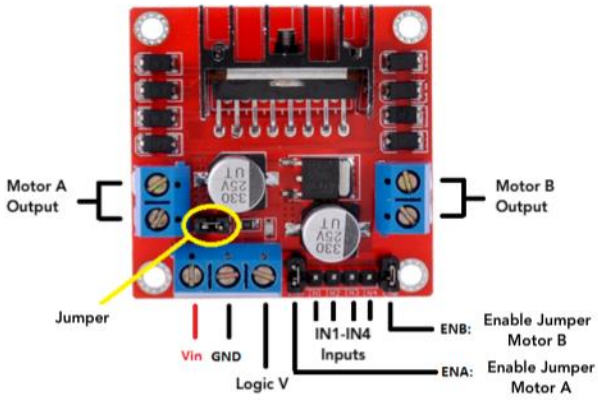


FIGURE 25 - SCHEMATIC OF THE L298N. RETRIEVED FROM: HTTPS://WWW.PROMETEC.NET/L298N/

The Peltier cell is directly plugged to the MotorA Output and the jumper is taken out to be able to supply the driver with a PWM signal. The PWM signal allows to adjust the current supplied to the Peltier cell and make the system work at different rates:

- High PWM signal Duty Cycle -> the system works faster
- Low PWM signal Duty Cycle - > the system works slower

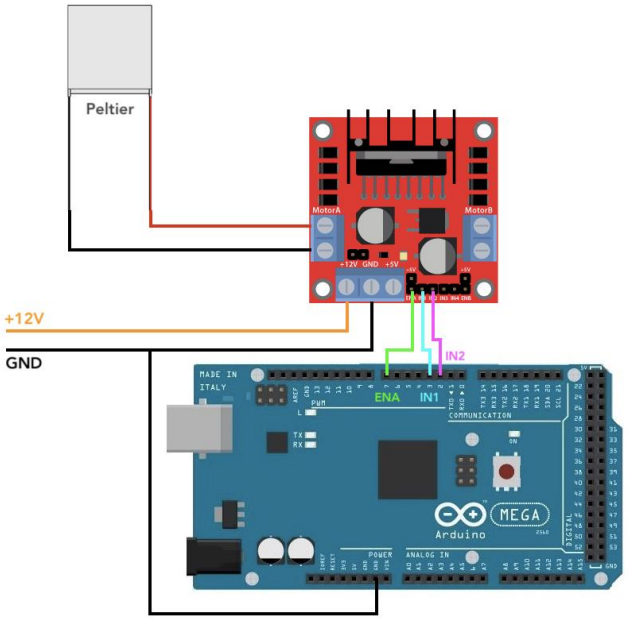


FIGURE 26 - SYSTEM USED TO TRY THE L298N WITH THE PELTIER CELL

With the setup in Figure 26, simple control of the Peltier cell can be achieved by playing with two variables:

ENA (PWM signal): With the PWM-enable signal the current supplied to Motor A/Peltier cell can be controlled. Using the Duty Cycle (DC %) of this signal the system can be put to work faster or slower, as explained before.

IN1 & IN2: With these two inputs the direction of the current can be changed. By changing the direction, the Peltier cell can be put in heating or cooling mode. Two functions were then created: heat() and cool().

TABLE V
HEATING/COOLING MODES

Mode/function	IN1	IN2
Heat	High	Low
Cool	Low	High

The following experimental results were obtained when varying the two variables. The green signal is the MotorA Output and the yellow signal corresponds to the PWM signal Input to the H-Bridge from the Arduino. As it can be seen in Figure 27, once the system cools down to 14-15 ºC it cannot reach lower temperatures and so it stabilizes. A heat sink had to be installed below the Peltier cells to help the system cool.

FIGURE 27 - YELLOW: H-BRIDGE PWM INPUT SIGNAL DC=0,75. GREEN: H-BRIDGE MOTOR A OUTPUT. SCALE: 2V/500µS.

With a Duty Cycle of 75% the Peltier cell cools down at a fast rate to down to 15 ºC. With a DC of 75% the system cannot reach temperatures lowers than 15 ºC, once this temperature is reached it stabilizes. But in order for the system to stay steady at 15 ºC, the DC must stay at 75% or higher; if the DC is lowered then the temperature rises again (Figure 28).

FIGURE 28 - YELLOW: H-BRIDGE PWM INPUT SIGNAL DC=0,25. GREEN: H-BRIDGE MOTOR A OUTPUT. SCALE: 2V/500µS.

As said before, in Figure 28 the DC was changed to 25% once the system reached 15 ºC, and so it began to heat. Finally, with a DC of 100% the system can the lowest temperature of 14 ºC. As the Peltier cell is working at its maximum speed the system reaches this temperature really fast. Fourteen is the lowest temperature the system is able to achieve.

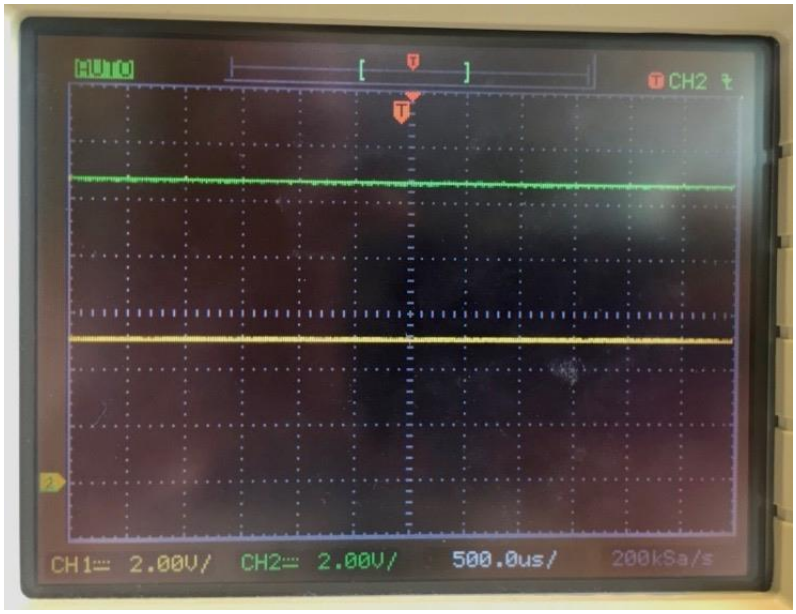


FIGURE 29 - YELLOW: H-BRIDGE PWM INPUT SIGNAL DC=1,00. GREEN: H-BRIDGE MOTOR A OUTPUT. SCALE: 2V/500µS.

7.4. PI CONTROL

As said before, to control the temperature a Peltier is used, to supply the Peltier a driver is needed and to control the driver a PI control is needed. To control the driver the Arduino plays with two variables: PWM's DC and the cooling/heating mode.

7.4.1. PI CODE

The goal of the control was to achieve and maintain a set temperature by acting on the H-Bridge + Peltier system. The chosen control was a PI, where the proportional part gives an output proportional to the current error and the integral part gives the necessary action to erase the steady-state error.

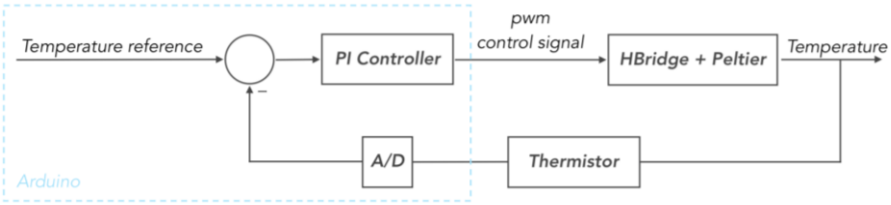


FIGURE 30 – PI CONTROL LOOP

Before entering the H-Bridge, the control signal must be multiplied by a factor, denominated pwm_factor. This way the control Kp and Ki parameters can be low numbers.

$$pwm\ control\ signal = output\ PI\ Controller\ x\ pwm\_factor$$

EQ. 7

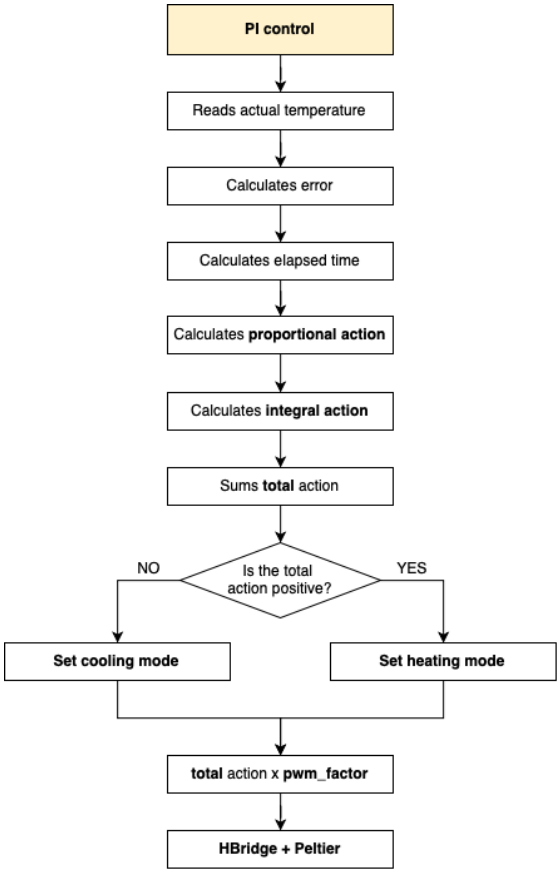


FIGURE 31 – PI CONTROL FLOWCHART

7.4.2. Ziegler-Nichols closed-loop method

The method used to adjust the Kp and Ki parameters was the Ziegler-Nichols method [22], this method allows the adjustment of the PI parameters without knowing the system's or plant's equations.

The procedure is as follows:

1st- Make the integral and proportional actions equal to zero (Ki=Kp=0).

2nd- Slowly increase the proportional action until the system oscillates, even when facing perturbations, in a stable manner

3rd- The proportional gain at that moment becomes the critical gain (Kc=Kp) of the system, and the oscillation period the critical period (T=Tc).

4th- Calculate the proportional and integral gains as shown in Table VITable .

Table VI
Ziegler-Nichols PI parameters calculation

Kp	Ki
0.45 * Kc	0.54*Kc/Tc

7.4.3. Ziegler-Nichols experimental procedure

To obtain the PI parameters following the Ziegler-Nichols method the setting from Figure 23 was used. In order to simulate a step input, the system was first cooled to 20 ºC, once that temperature was reached it was then programmed to reach 25 ºC.

As it is explained before, the control signal that exists the closed control loop must be multiplied by a "pwm_factor" before entering the H-Bridge. After trying multiple factors, the ones chosen for the experiments were pwm_factor=500, 700 and 900.

For each pwm_factor the Ziegler-Nichols procedure was carried out and as a result, three different controllers were obtained, one for each pwm_factor (see ANNEX 14.2 "Bill of materials" for more detail on this matter).

The gains for each controller can be seen on this table:

Table VII
Calculated PI parameters for each controller

pwm_factor	Kp	Ki
500	0.126	0.016
700	0.315	0.038
900	0.383	0.046

To choose the appropriate control the experimental procedure was carried out again, but this time with the PI parameters for each controller. The temperature response of the system was monitored and recorded for each controller. Finally, all of them were compared and the final controller was chosen. In order to choose the best controller, the following was taken into account: settling time, overshoot and final error.

CASE: pwm_factor=500

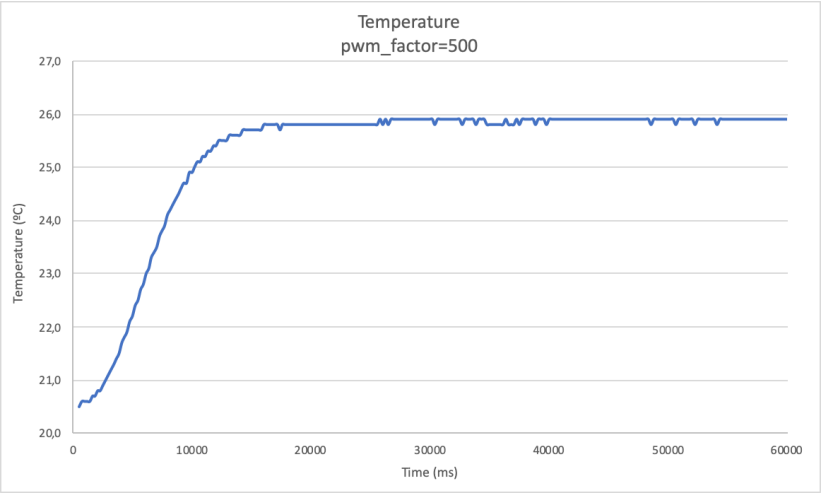


FIGURE 32 - TEMPERATURE RESPONSE FOR A PWM_FACTOR OF 500

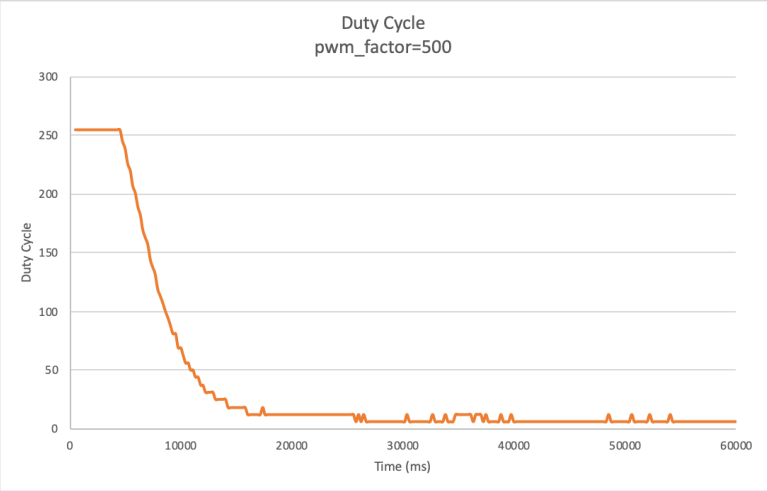


FIGURE 33 - DUTY CYCLE OF THE PWM INPUT SIGNAL TO THE H-BRIDGE FOR A PWM_FACTOR OF 500

The temperature response of the system was fast, but the final temperature always had some error with respect to the reference temperature. The settling time (1%) for this controller is 15 seconds, it has no overshoot and always has an error of 0,2-0,1 ºC with respect to the set reference temperature (0,77-0,38% error).

CASE: pwm_factor=700

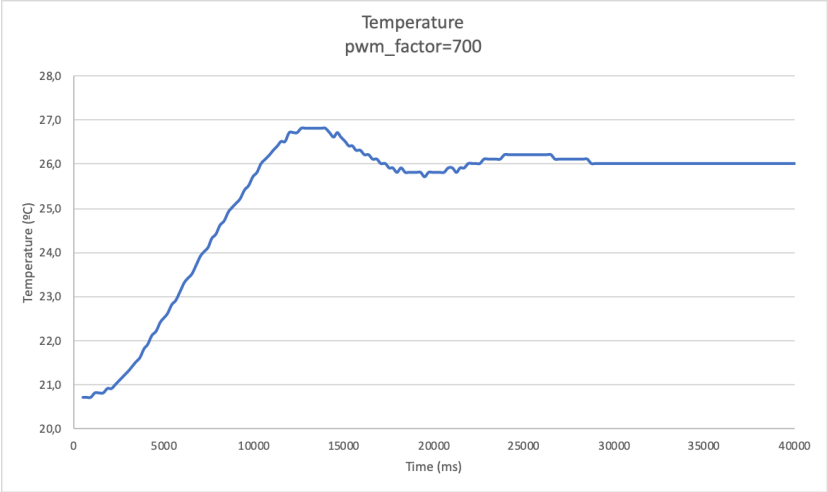


FIGURE 34 - TEMPERATURE RESPONSE FOR A PWM_FACTOR OF 700

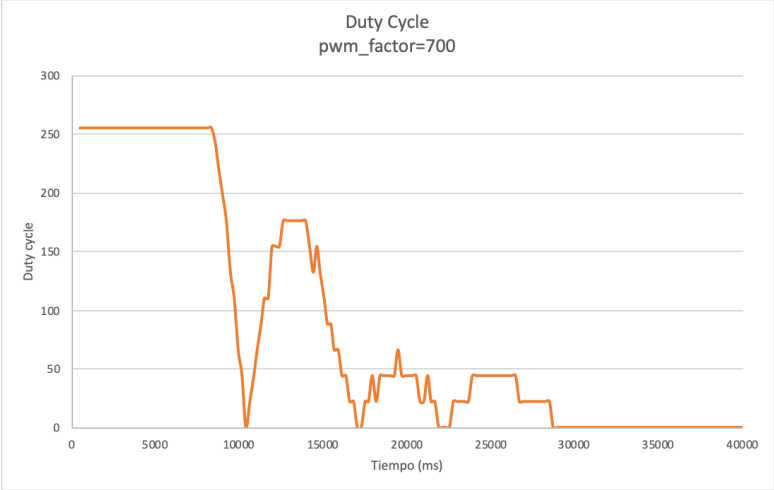


FIGURE 35 - DUTY CYCLE OF THE PWM INPUT SIGNAL TO THE H-BRIDGE FOR A PWM_FACTOR OF 700

The temperature response of the system was a little bit slower than in the first case, but the final temperature had no error. The settling time (1%) for this controller is 19 seconds, it has a 25,8% overshoot and no difference with respect to the set temperature.

CASE: pwm_factor=900

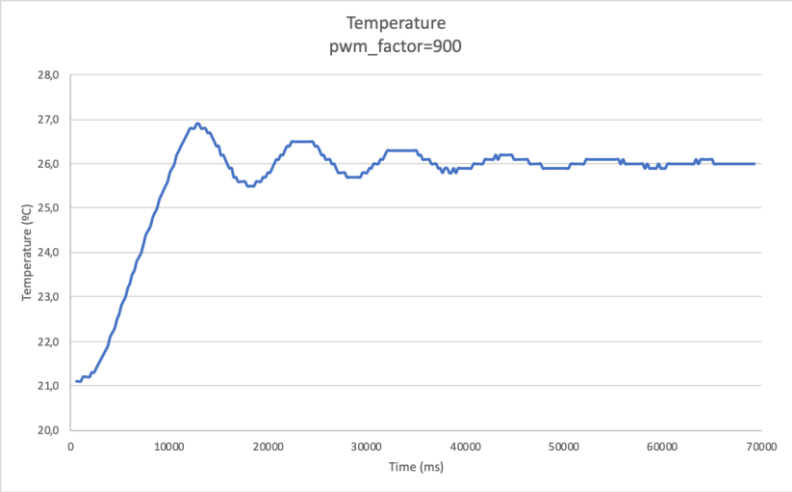


FIGURE 36 - TEMPERATURE RESPONSE FOR A PWM_FACTOR OF 900

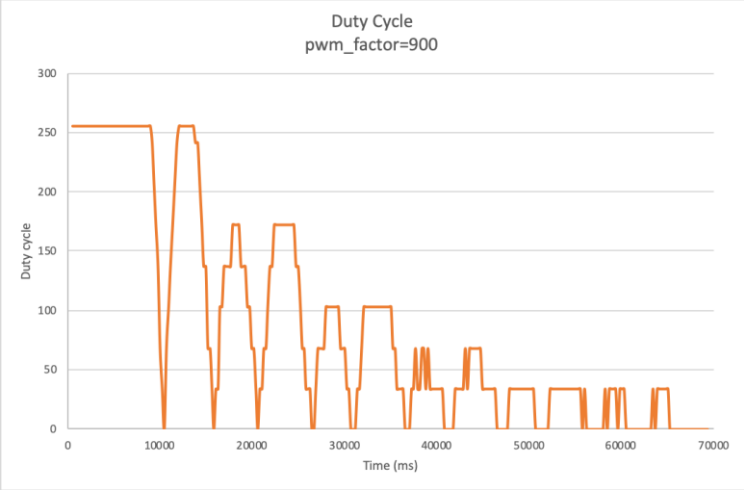


FIGURE 37 - DUTY CYCLE OF THE PWM INPUT SIGNAL TO THE H-BRIDGE FOR A PWM_FACTOR OF 900

The temperature response of the system was the slowest and the final temperature had again a 0,2-0,1 ºC error (0,77-0,38%). The settling time (1%) for this controller is 29 seconds, it has a 25,9% overshoot and an error of less than one percent.

TABLE VIII
SUMMARY OF THE DIFFERENT SYSTEM'S RESPONSES

pwm_factor	Kp	Ki	ts (1%)	Overshoot	error (%)
500	0.126	0.016	15	–	0.77-0.38
700	0.315	0.038	19	25.8	0
900	0.383	0.046	29	25.9	0.38

The final controller chosen was the one with the pwm_factor of 700, it has a good settling time and has no error or a very large overshoot.

7.4.4. FINAL CONTROL ADJUSTMENTS

When the final controller was tested, the final temperature had errors with respect to the desired one when dealing with big temperature changes (from 20ºC to 30ºC for example). In order to solve this problem, a new variable was introduced. This variable deactivates the integral action when a new desired temperature is introduced. Once the temperature reaches a value ±2ºC from the desired one, the integral part is reactivated.

8. EXPERIMENTAL PROCEDURE – FINAL SYSTEM ASSEMBLY

Once all parts (XYZ movement and temperature control) had been designed, tested and successfully programmed, it was time to merge both systems into one final system and all the code into one final file.

As explained before, the goal was for the system to:

- Read CNC commands and act accordingly
- Control XY movement (stepper motors)
- Control Z movement (servo motor)
- Control temperature system

Also, the code was divided into ".h" files, with each one containing a different part of the code. This makes the main code easily compatible with new code additions or makes easier to do not include parts of the code if these are not going to be used.

Once the final system and final code were ready, the code was transferred into the Arduino, and the entire system was tested. Everything proved to work perfectly.

NOTE: The code flowcharts and the entire code can be found on the ANNEX (14.4 p.59, 14.5 p.67)

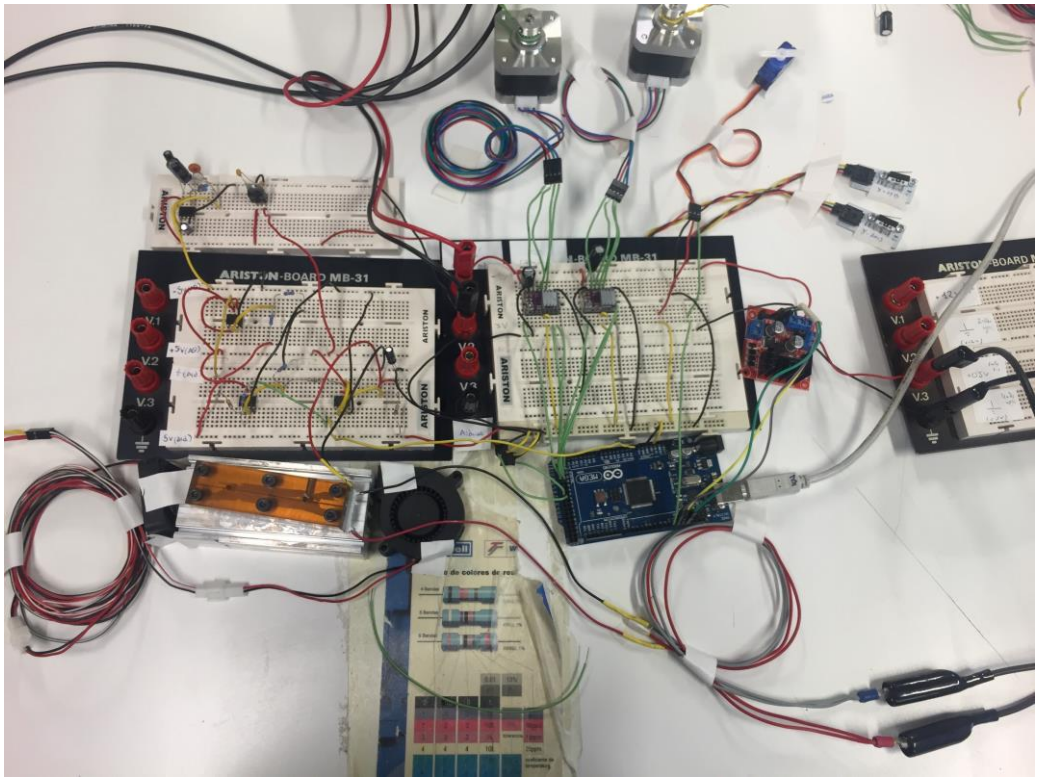
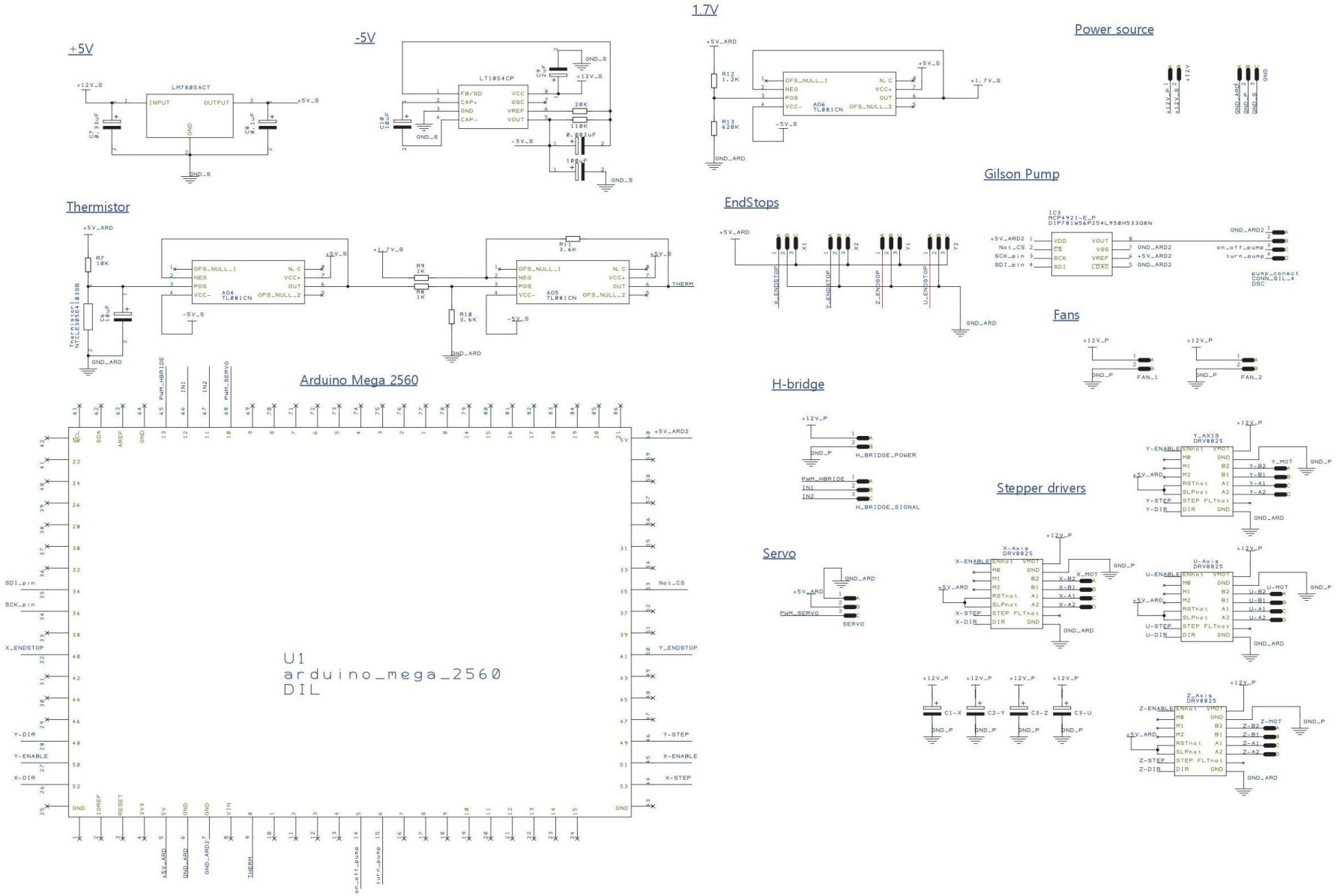


FIGURE 38 - UNIFICATION OF BOTH SYSTEMS IN PROTOBOARD

9. Electronic design – Arduino Mega shield

All of the components mentioned in the sections above (Motor and Temperature control) are integrated into an Arduino Mega shield.



9.1. ARDUINO MEGA PINOUT

The schematic in Figure 39 symbolizes the Arduino Mega 2560 pinout.

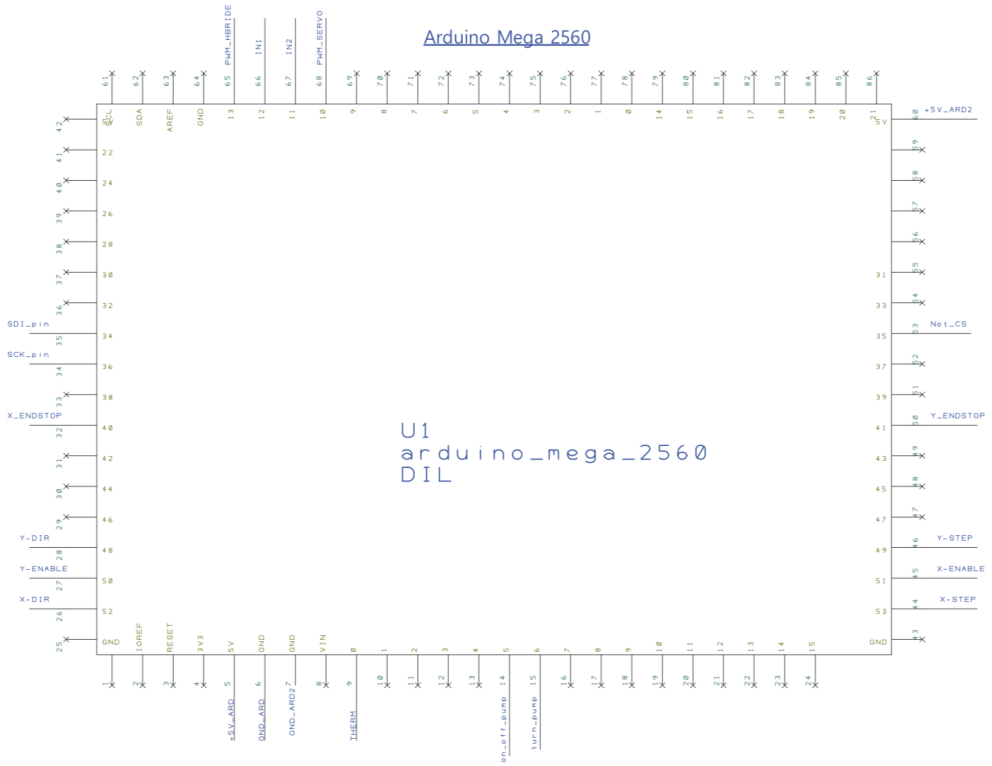


FIGURE 39 - ARDUINO MEGA SCHEMATIC

In the next figure (Figure 40) the connections used are located in the Arduino.

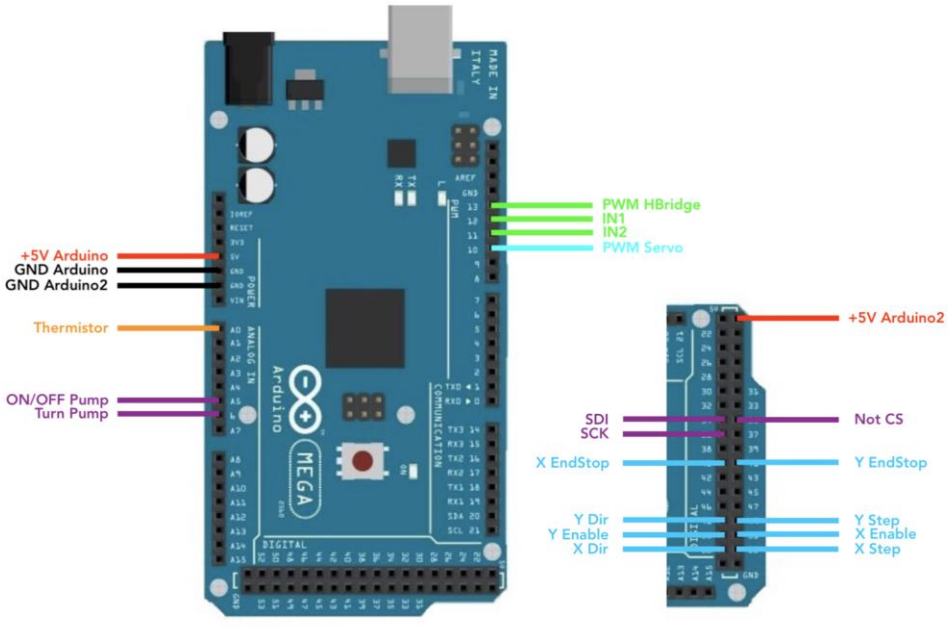


FIGURE 40 - ARDUINO MEGA REAL PINOUT AND CONNECTIONS

9.1. POWER SUPPLY

In the design of the shield the power and digital/sensor signals are separated, sensor signals are much weaker than power signals and must be protected.

The shield has been designed so it only has one input power supply or power source. The calculated power source should supply +12 V and a maximum current of 2 A, according to the final verifications of the overall system.

The +12 V volts from the power source are then distributed, in parallel, to the circuits dealing with power signals and to the circuit that generates the ± 5V used in the Operational Amplifiers. All of the ground signals within the shield are connected, in parallel, to the +12 V power supply as well.

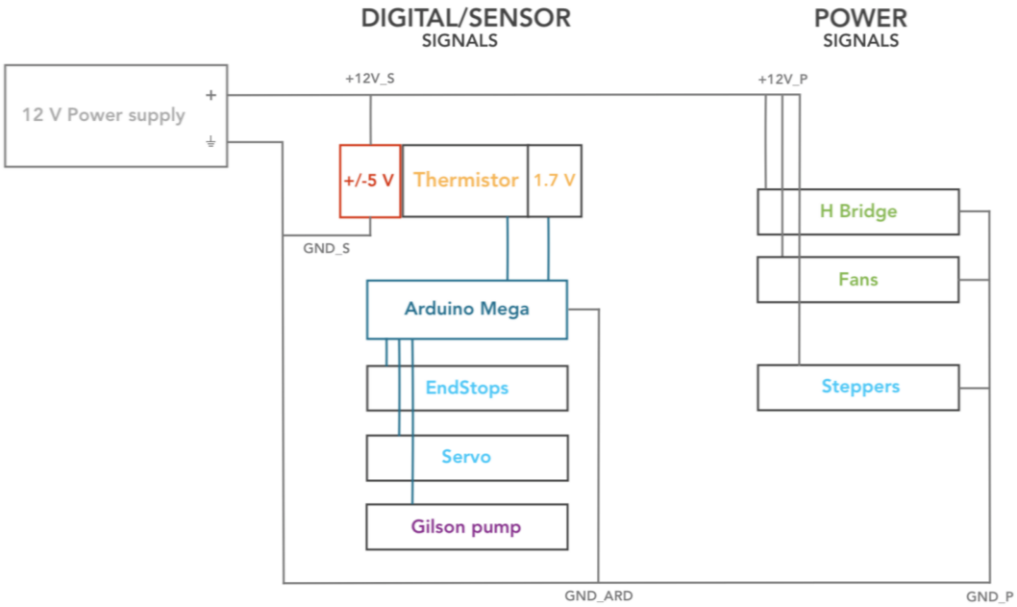


FIGURE 41 - DISTRIBUTION OF +12 V AND GROUND SIGNALS

TABLE IX
DISTRIBUTION OF +12 V AND GROUND SIGNALS

	Power supply	Ground
Digital/Sensor signals	+12V_S +5V_ARD	GND_S GND_ARD
Power signals	+12V_P	GND_P

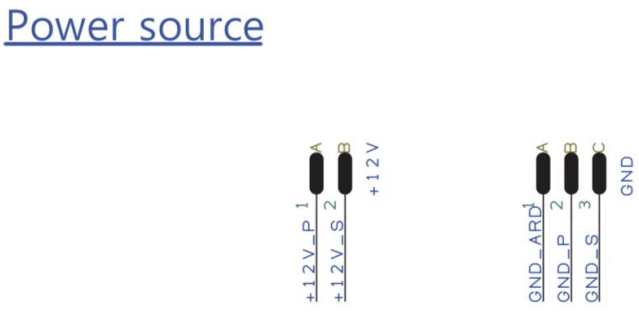


FIGURE 42 - POWER SOURCE SCHEMATIC

9.2. POWER SIGNALS

9.2.1. H BRIDGE – PELTIER CELLS

The connections for the H-Bridge that supplies power to the Peltier cell is shown in Figure 43. The H-Bridge is powered by the +12 V power source and has its ground as well, it is also connected to a PWM Arduino signal (PWM_HBRIDGE) and two digital ones (IN1 & IN2).

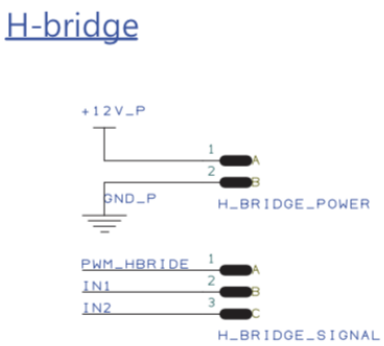


FIGURE 43 - H-BRIDGE CONNECTION

9.2.2. FANS

The fans used in the Temperature control system are also connected to the +12V power source and to its ground.

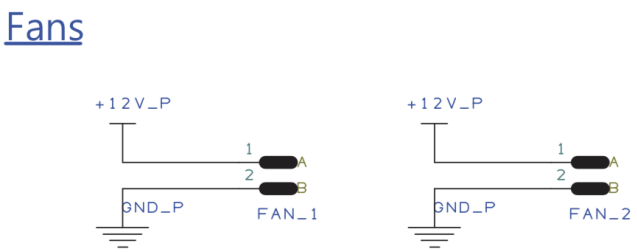


FIGURE 44 - FANS CONNECTIONS

9.2.3. STEPPERS

The connections for the Stepper drivers can be found in the figure below. The drivers for the Steppers have a part powered by the +12V power source, with its ground as well; and another part powered by the +5V from the Arduino, with its ground. The control of each of the drivers is carried out with their corresponding Arduino digital signals (X-ENABLE, X-STEP, X-DIR…).

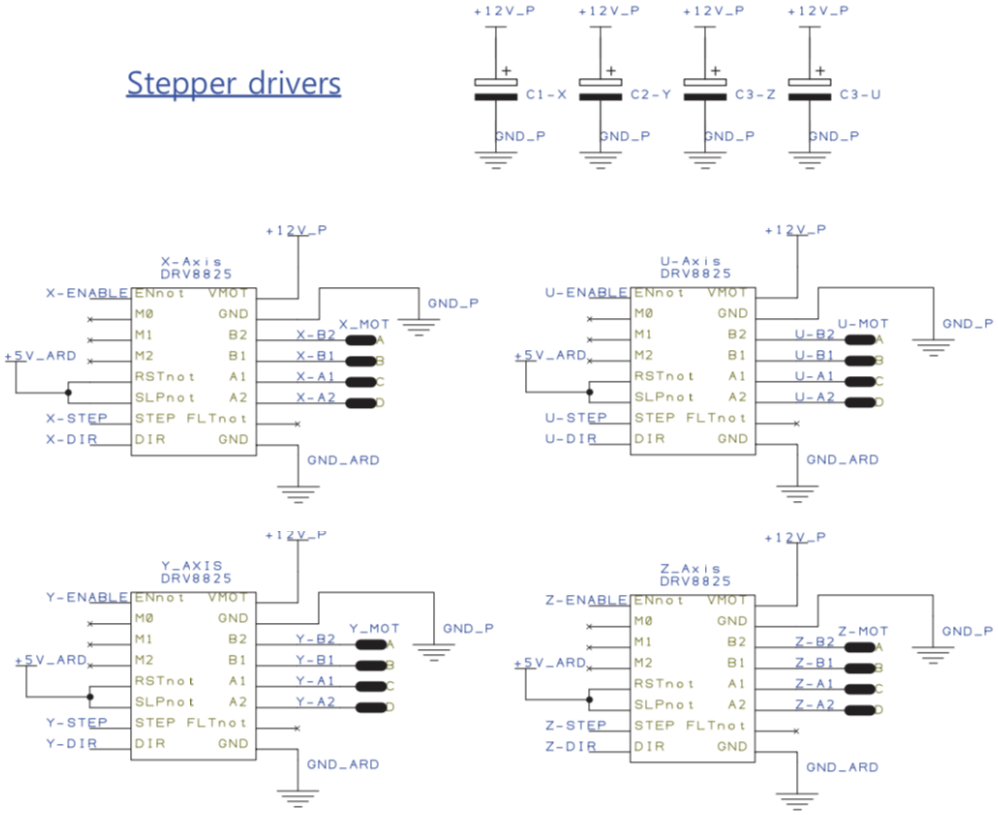


FIGURE 45 - STEPPER DRIVER'S CIRCUIT

9.3. DIGITAL/SENSOR SIGNALS

9.3.1. ±5 V SUPPLY

The ±5 V are used as the power supply for all operational amplifiers used in the shield.

+5V

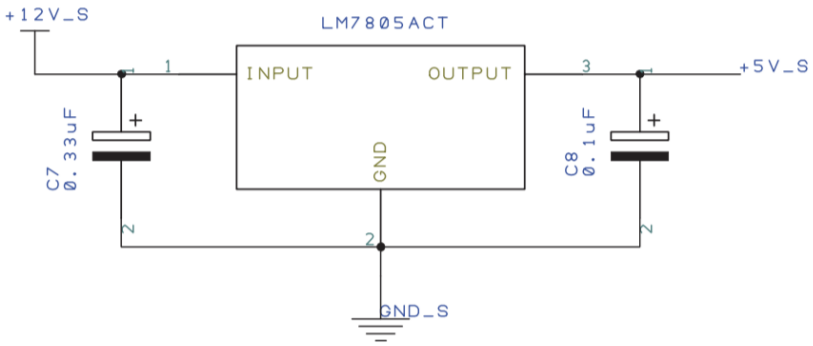


FIGURE 46 - +5 V POWER SUPPLY CIRCUIT

A +5 V power supply was designed to have a constant and stable positive five voltage source (Figure 46). The circuit is built of a 3 terminal 1A Positive Voltage Regulator (LM7805ACT), see [23], a +12 V power source, and a couple of capacitors C7(330 nF) and C8(0.1 µF).

-5V

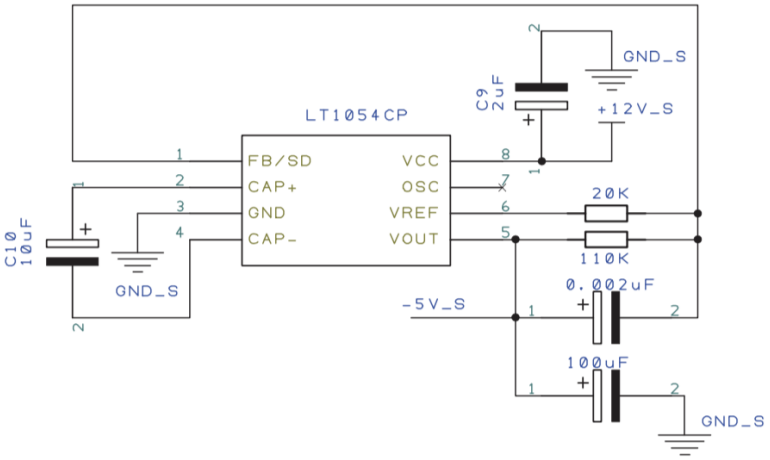


FIGURE 47 - -5 V POWER SUPPLY CIRCUIT

The -5 V power source circuit (Figure 47) is built with a switched-capacitor voltage converter (LT1054CP), see [24], a +12 V power source, four capacitors C9(2 µF), C10(10 µF), C(2 nF) and C(100 µF), and a couple of resistances R(20 kΩ) and R(110 kΩ).

9.3.2. +1.7 V SOURCE

As it was explained in section 7.2.2 (p.20) the thermistor's circuit needs a 1.7V source to subtract it to the signal from the sensor. As seen in Figure 48 the voltage is obtained from a voltage divider using the Arduino's 5V and a couple of resistances R12(1.2 kΩ) and R13(620 kΩ); also, an operational amplifier is used as a voltage follower to have a constant 1.7V as an output.

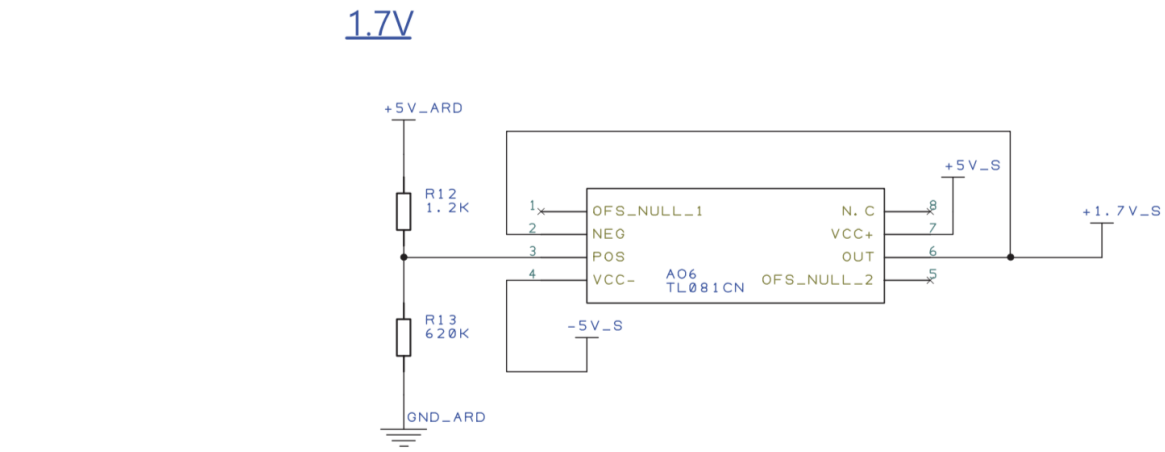


FIGURE 48 - 1.7V GENERATOR

9.3.3. THERMISTOR

The thermistor's circuit for the shield is shown in Figure 49. As explained before this circuit is composed of two operational amplifiers, five resistors, one capacitor and a thermistor. As power sources, it uses the +5 V from the Arduino, and its ground, and the ±5 V for the OA. This circuit also uses the +1.7 V source.

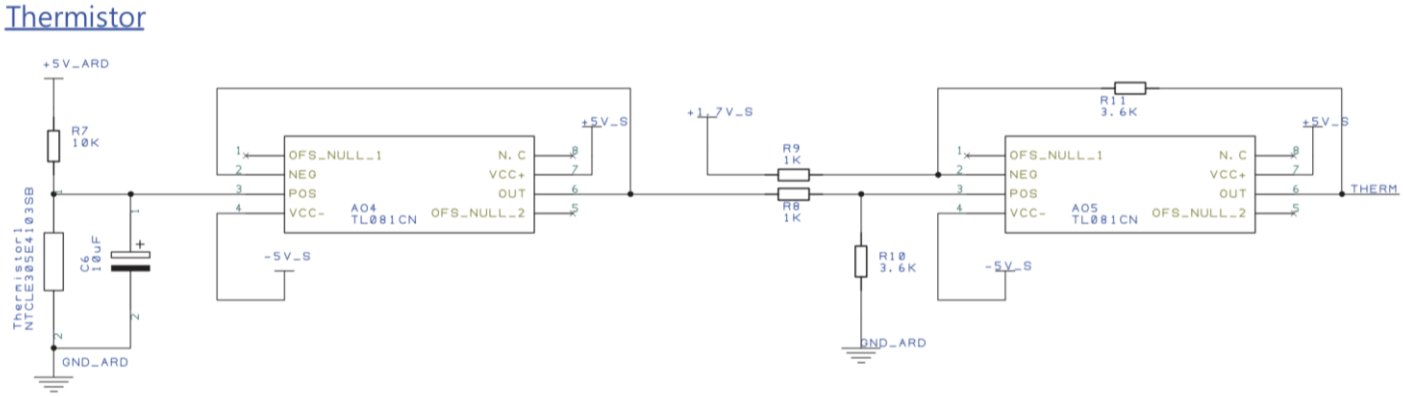


FIGURE 49 - THERMISTOR'S CIRCUIT

9.3.4. ENDSTOPS

The Endstops are also part of the sensor signals. They are supplied by the Arduino's +5V and their signal goes to the Arduino Mega's digital inputs.

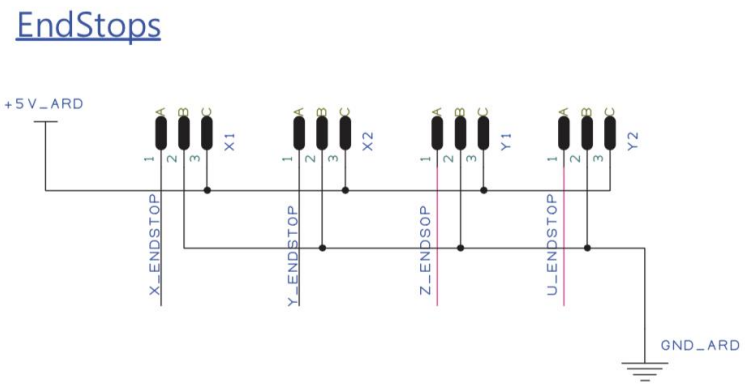


FIGURE 50 - ENDSTOPS CIRCUIT

9.3.5. SERVO

The Servo motor used is also supplied by the +5V from the Arduino and connected to its ground. The signal that controls it is a PWM Arduino signal (PWM_SERVO).

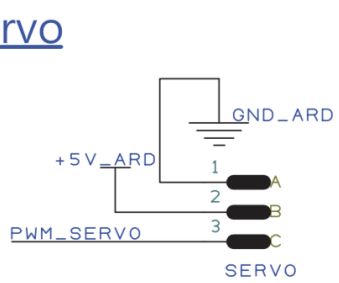


FIGURE 51 - SERVO CONNECTIONS

9.3.1. GILSON PUMP

The connections for the Gilson Pump circuit are shown in Figure 52. The D/A converter is powered by the +5V Arduino signal and its ground, and it is controlled with Arduino digital signals (Not_CS, SCK_pin and SDI_pin). The connection to the pump is also shown in the figure.

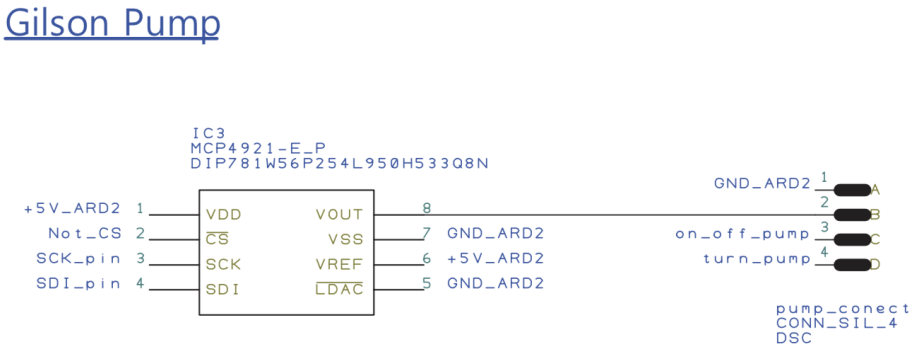


FIGURE 52 - GILSON PUMP CIRCUIT

9.4. PCB

The final PCB layout can be seen in Figure 55. As said before it is a shield for an Arduino Mega board and it has all the circuits shown in the previous sections.

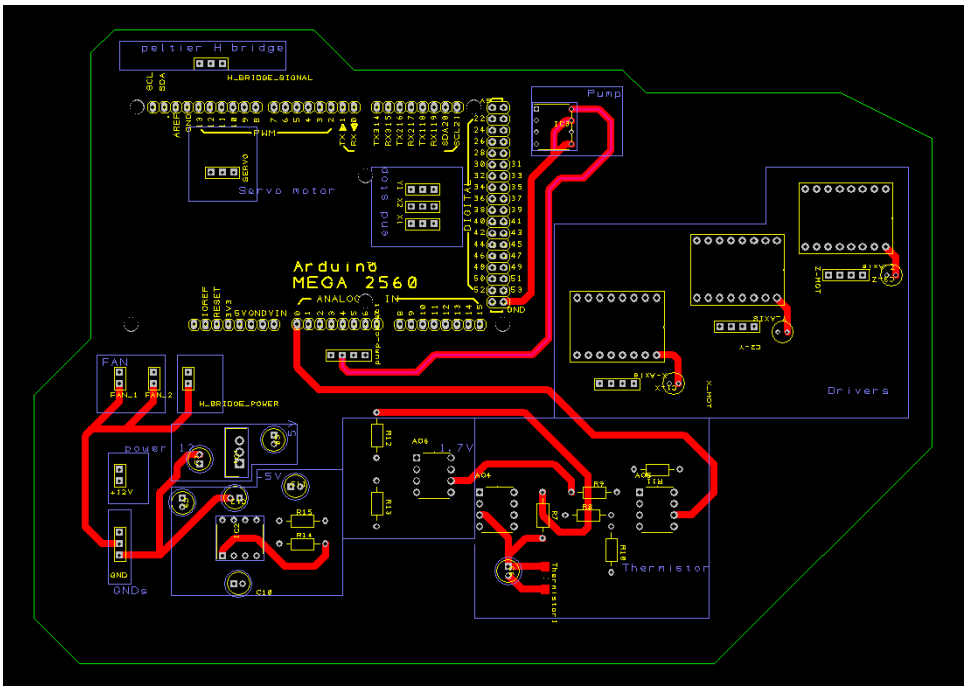


FIGURE 53 - TOP PCB LAYER

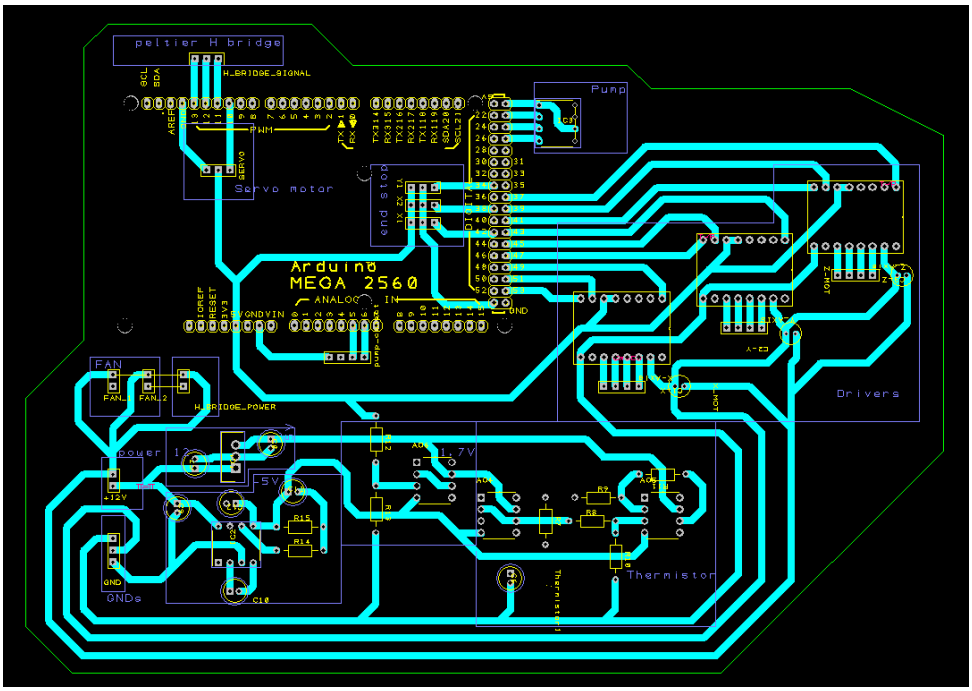


FIGURE 54 - BOTTOM PCB LAYER

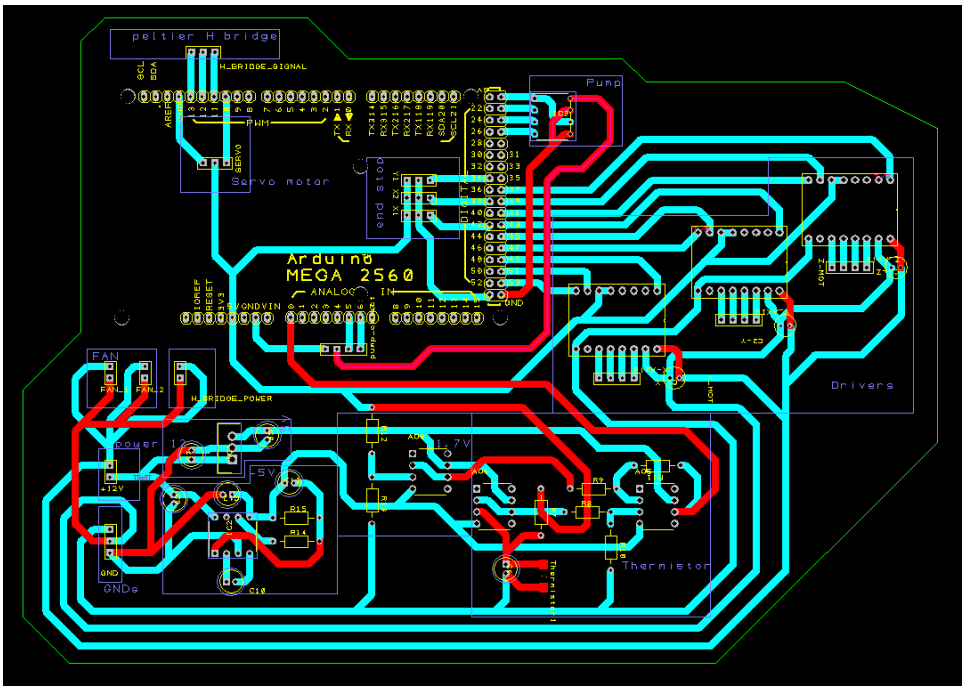


FIGURE 55 - TOP AND BOTTOM PCB LAYERS

10. GRAPHICAL USER INTERFACE

The system has been coded so the user can move and program the system with a Raspberry touchscreen using CNC commands. Those commands are then read by the Arduino and then it performs the given action.

A few changes have been made to the original GUI programmed in this thesis [25].

10.1. CONFIGURATION

This is the main window of the GUI, where the user can configure the robot.

Serial
In this area, the user can select the port for the serial communication and the baud rate (set to 115200 in the main code).

File
Here the user can load the configuration of the position of each vial/sample in the rectangular matrix.

Status
In these two displays, the user can check the flow cell and speed suction set. Also can enable temperature check.

Control
Here the user can control the movement of the robot, change the PI parameters or set a new temperature of reference.
The user can choose to move each axis a given distance in mm, do homing or release the motors so they can be moved manually. The PI parameters are set by default as mentioned earlier, but the user can choose to change them. Also, the desired temperature of the system can also be changed.

Console
In this area the user can see the commands the Arduino is executing at that time and also can send commands to the Arduino as well.

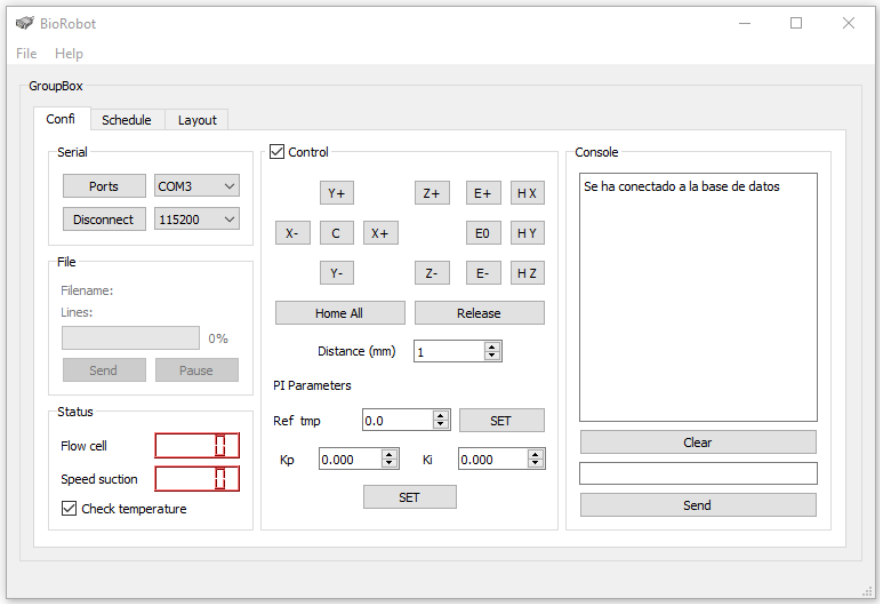


FIGURE 56 - CONFIGURATION WINDOW

10.2. SCHEDULE

This is the window of the GUI where the programmed vials/samples are displayed. Each position matches one sample, with its own speed and time of suction. The user can choose to enable or disable the suction of the position.

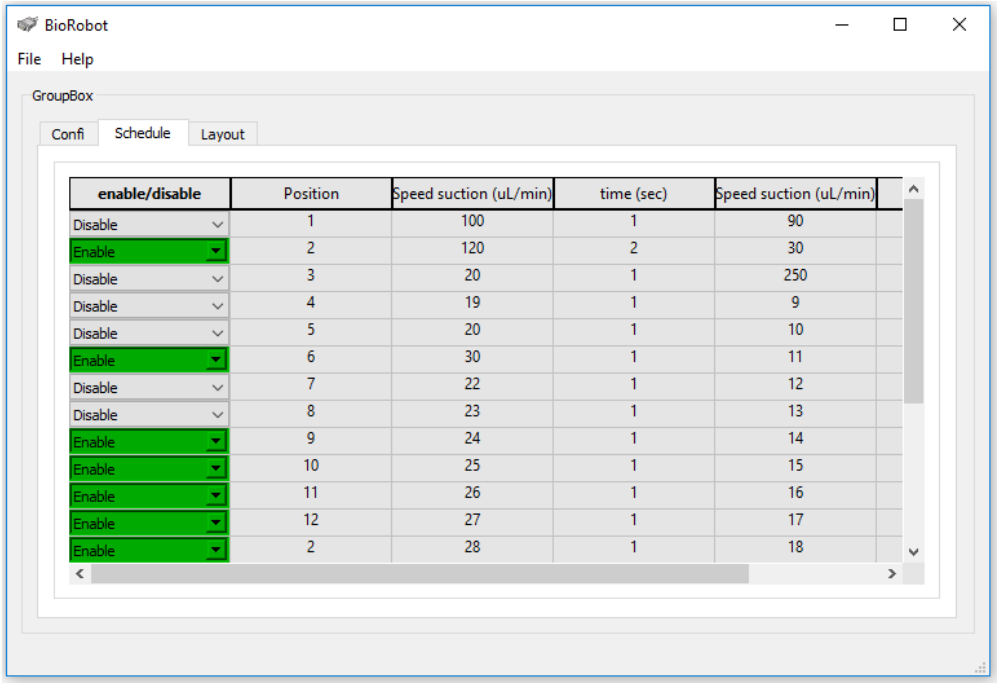


FIGURE 57 - SCHEDULE WINDOW

10.3. LAYOUT

Once the Schedule window is programmed, the user can check the layout of the different samples selected with the layout window.

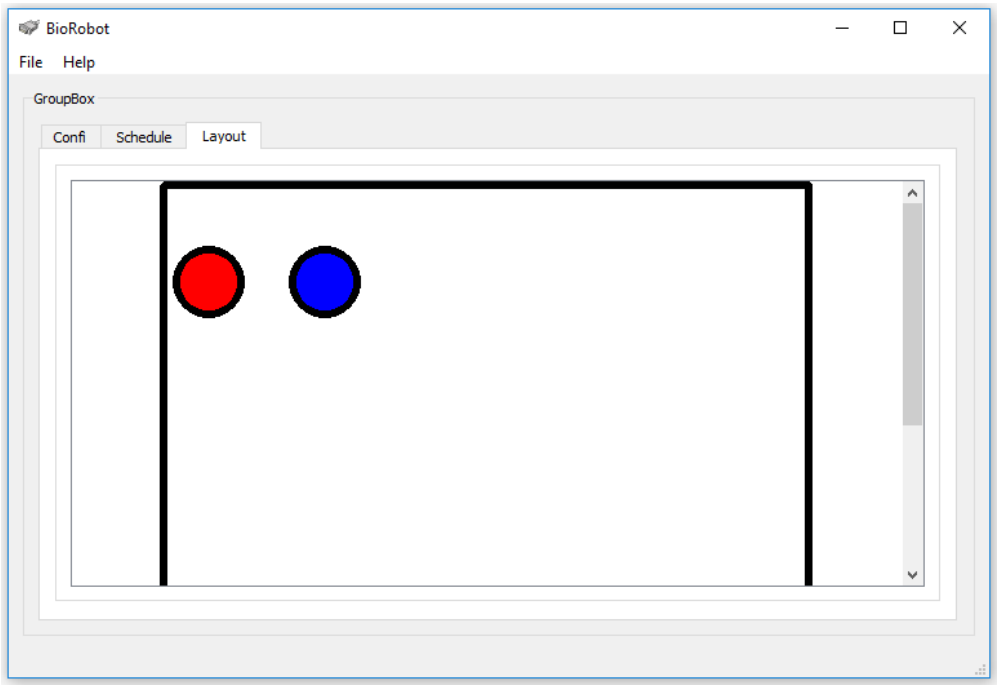


FIGURE 58 - LAYOUT WINDOW

11. CONCLUSIONS

The main objective of this project was to design and elaborate a Cartesian automated system for the fabrication of biosensors, focusing on its movement and temperature control. All the necessary studies and experimental procedures for the development of the two systems have been successfully accomplished. Both the movement and temperature control systems have been designed, programmed and finally integrated into one final system.

Knowledge of electronic design, gained throughout the degree in industrial engineering, has been key in the electronic design of the circuits. This part of the project also involved new electronic components, which required getting to know how to implement and work with them. The design of the printed circuit board (PCB) meant learning to work with DesignSpark.

Another goal was to make a versatile code, able to easily work with different hardware components or code files. As a result, this part has also meant a considerable improvement in C++ programming skills.

12. FUTURE RESEARCH LINES

There are some experimental procedures still in progress. The welding of the electronic components into the printed PCB, see Figure 55, will be done once the printed PCB is ready. Also, the mechanical structure of the system is being 3D printed and built, see Figure 59.

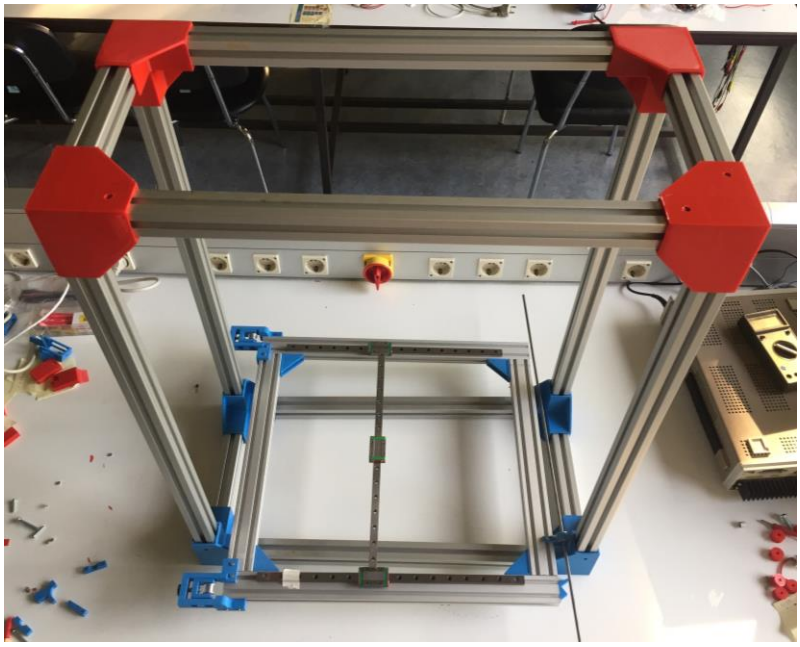


FIGURE 59 - MECHANICAL STRUCTURE IN PROGRESS

This versatile system could also be studied to be used as a layer-by-layer assembly system. These systems alternate layers of oppositely charged materials and wash steps in between [26].

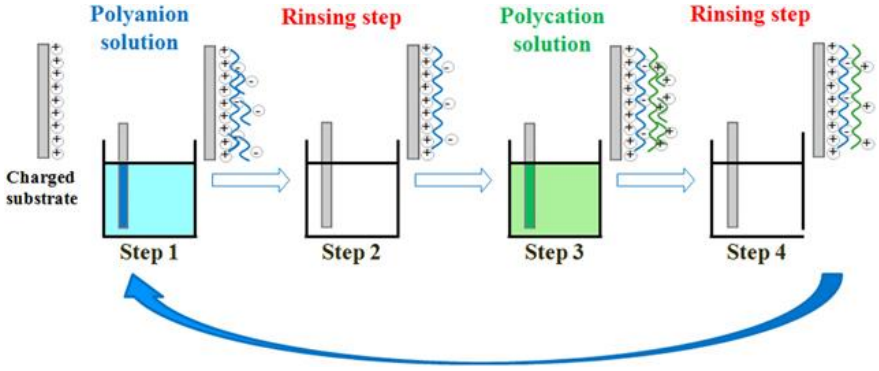


FIGURE 60 - LBL ASSEMBLY TECHNIQUE

13. BIBLIOGRAPHY

[1] M. P. Groover, "Automation | Britannica.com," *Encyclopedia Britannica*, 2002. [Online]. Available: https://www.britannica.com/technology/automation.

[2] H. co. Editors, "Industrial Revolution," *History*, 2019. [Online]. Available: https://www.history.com/topics/industrial-revolution/industrial-revolution.

[3] C. co. Editors, "The History of Computer Numerical Control (CNC)," *CNC*. [Online]. Available: http://www.cnc.com/the-history-of-computer-numerical-control-cnc/.

[4] K. Olsen, "The First 110 Years of Laboratory Automation," *J. Lab. Autom.*, vol. 17, no. 6, pp. 469–480, Dec. 2012.

[5] D. Goldberg, "History of 3D Printing: It's Older Than You Think," *Redshift by Autodesk*, 2018. [Online]. Available: https://www.autodesk.com/redshift/history-of-3d-printing/.

[6] D. Workshop, "Stepper Motors with Arduino - Getting Started with Stepper Motors," 2018. [Online]. Available: https://dronebotworkshop.com/stepper-motors-with-arduino/.

[7] Orientalmotor, "Stepper Motors, Stepper Motor Drivers and Controllers." .

[8] J. Loureiro, "La comparativa definitiva de los drivers para motores paso a paso : DRV8825 vs A4988," *Staticboards*, 2016. .

[9] Pololu, "Pololu - DRV8825 Stepper Motor Driver Carrier, High Current (md20a)." [Online]. Available: https://www.pololu.com/product/2133. [Accessed: 20-Feb-2019].

[10] Prometec, "Conociendo los servos," *Prometec*. [Online]. Available: https://www.prometec.net/servos/.

[11] W. Electronics, "Thermistor basics." [Online]. Available: https://www.teamwavelength.com/thermistor-basics/.

[12] M. del Campo García, "Generar frío con una célula Peltier cerámica TEC1-12706," *Mi Arduino Uno tiene un blog*, 2016. .

[13] Prometec, "EL MÓDULO CONTROLADOR DE MOTORES L298N," *Prometec*. .

[14] Stepperonline, "Nema 17 Bipolar 45Ncm (64oz.in) 2A 42x42x40mm 4 Wires w/ 1m Cable & Connector." .

[15] SparkFun Electronics, "Level Up Your Arduino Code: External Interrupts," *Youtube*, 2018. [Online]. Available: https://www.youtube.com/watch?v=J61_PKyWjxU&t=657s.

[16] D. Royer, "GcodeCNCDemo6AxisRumbaTimerInterruptV2." 2013.

[17] O. Beck, "GCodeAFMotorV2." 2014.

[18] M. Margolis and N. Bontrager, "ServoTimer2." 2013.

[19] Vishay, "NTCLE305E4 thermistor - Datasheet." pp. 1–5, 2017.

[20] Adaptative Thermal Management, "ET-131-10-13-S Peltier cooler module ET-131-10-13-S Peltier cooler module - Datasheet." pp. 1–4.

[21] E. van der Zalm, S. Lahteine, R. Neufeld, and B. Kuhn, "Marlin." 2018.

[22] C. Pardo, "Método de Ziegler-Nichols," 2019. [Online]. Available: https://www.picuino.com/es/arduprog/control-ziegler-nichols.html.

[23] Fairchild, "3-Terminal 1A Positive Voltage Regulator - Datasheet." pp. 1–23, 2005.

[24] Texas Instruments, "LT1054 Switched-Capacitor Voltage Converters With Regulators - Datasheet." Texas Instruments, pp. 1–35, 2015.

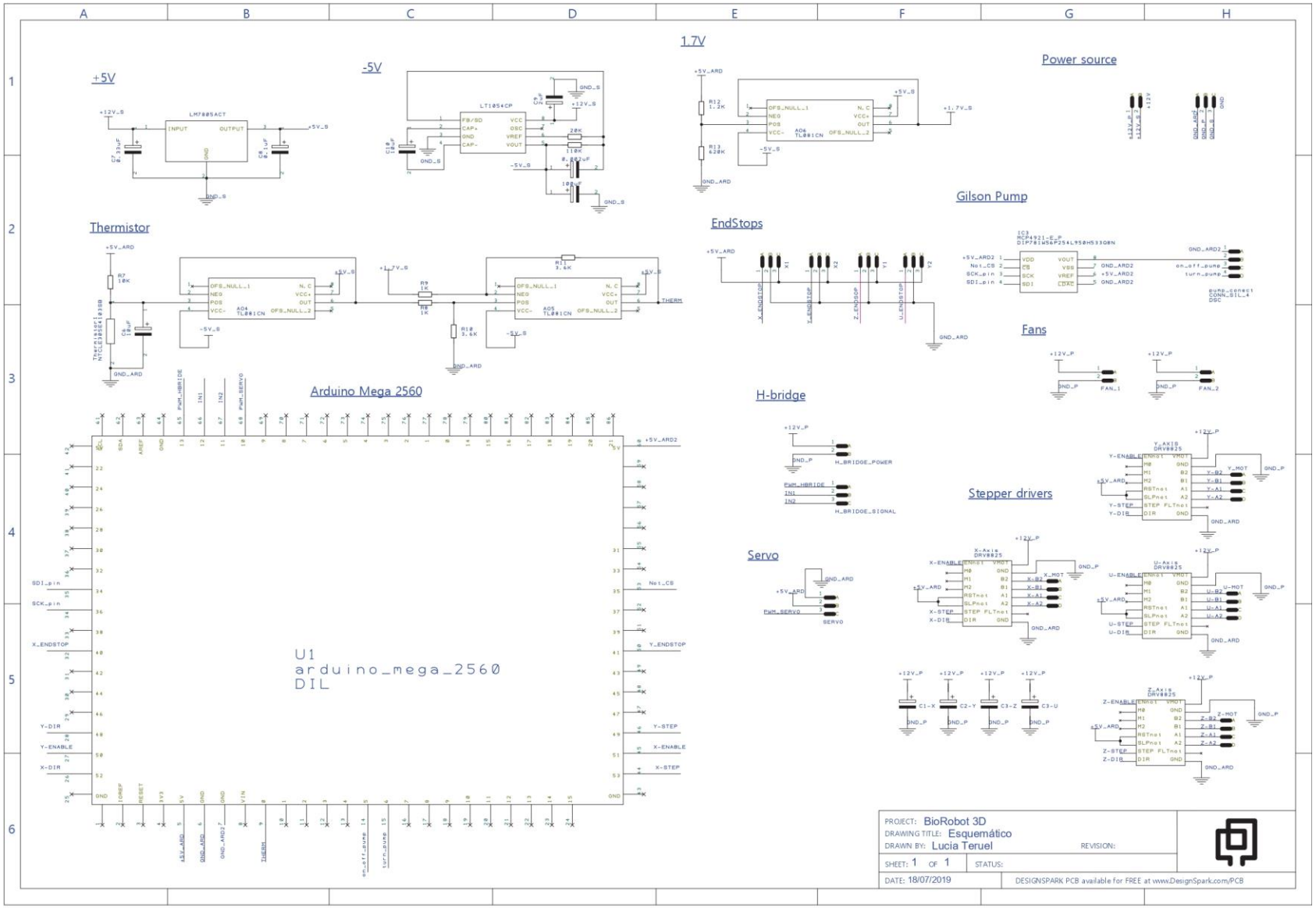
[25] V. G. Adrián, "Development of an Automated System Controlled by GUI for Experiments with

Optical Fiber Biosensors," Universidad Pública de Navarra, 2018.

[26] Wikipedia, "Layer by layer," *Wikipedia*, 2018. .

14. ANNEX

14.1. ARDUINO MEGA SHIELD



14.2. BILL OF MATERIALS

REF NAME	Qty	COMPONENT	VALUE	MANUFACT.	DESCRIPTION
U1		arduino_mega_2560			
H_BRIDGE_P	1	CONN_SIL_2			2 way Pin Header
FAN_1	1	CONN_SIL_2			2 way Pin Header
FAN_2	1	CONN_SIL_2			2 way Pin Header
+12V	1	CONN_SIL_2			2 way Pin Header
X1	1	CONN_SIL_3			3 way Pin Header
X2	1	CONN_SIL_3			3 way Pin Header
Y1	1	CONN_SIL_3			3 way Pin Header
SERVO	1	CONN_SIL_3			3 way Pin Header
H_BRIDGE_S	1	CONN_SIL_3			3 way Pin Header
GND	1	CONN_SIL_3			3 way Pin Header
X_MOT	1	CONN_SIL_4			4 way Pin Header
Y_MOT	1	CONN_SIL_4			4 way Pin Header
Z-MOT	1	CONN_SIL_4			4 way Pin Header
pump_conect	1	CONN_SIL_4			4 way Pin Header
X-Axis	1	DRV8825			
Y_AXIS	1	DRV8825			
Z_Axis	1	DRV8825			
IC1	1	LM7805ACT			Linear Voltage Regulator, 1A, 5 V 3-Pin
IC2	1	LT1054CP		Texas Instr.	Charge Pump Inverting, Step Up, -5 V, 8-Pin
IC3	1	MCP4921-E_P		Microchip	12 bit Serial DAC, 8-Pin PDIP
Thermistor1	1	NTCS0603E3103JMT		Vishay	RESISTOR, 0603 10K Ohms +/- 5% 125 mW
R7	1	Resistor	10K		Resistor
R8	1	Resistor	1K		Resistor
R9	1	Resistor	1K		Resistor
R10	1	Resistor	3.6K		Resistor
R11	1	Resistor	3.6K		Resistor
R12	1	Resistor	1.2K		Resistor
R13	1	Resistor	620K		Resistor
R14	1	Resistor	20K		Resistor
R15	1	Resistor	110K		Resistor
AO4	1	TL081CN		STMicroelectr.	JFET single operational amplifiers
AO5	1	TL081CN		STMicroelectr.	JFET single operational amplifiers
AO6	1	TL081CN		STMicroelect.	JFET single operational amplifiers
C6	1	UHE1H100MDD1TD	10µF	Nichicon	Aluminum Electrolytic Capacitors, Low Imped
C7	1	UHE1H100MDD1TD	330nF	Nichicon	Aluminum Electrolytic Capacitors, Low Imped
C8	1	UHE1H100MDD1TD	0.1µF	Nichicon	Aluminum Electrolytic Capacitors, Low Imped
C9	1	UHE1H100MDD1TD	2µF	Nichicon	Aluminum Electrolytic Capacitors, Low Imped
C10	1	UHE1H100MDD1TD	10µF	Nichicon	Aluminum Electrolytic Capacitors, Low Imped
C11	1	UHE1H100MDD1TD	2nF	Nichicon	Aluminum Electrolytic Capacitors, Low Imped

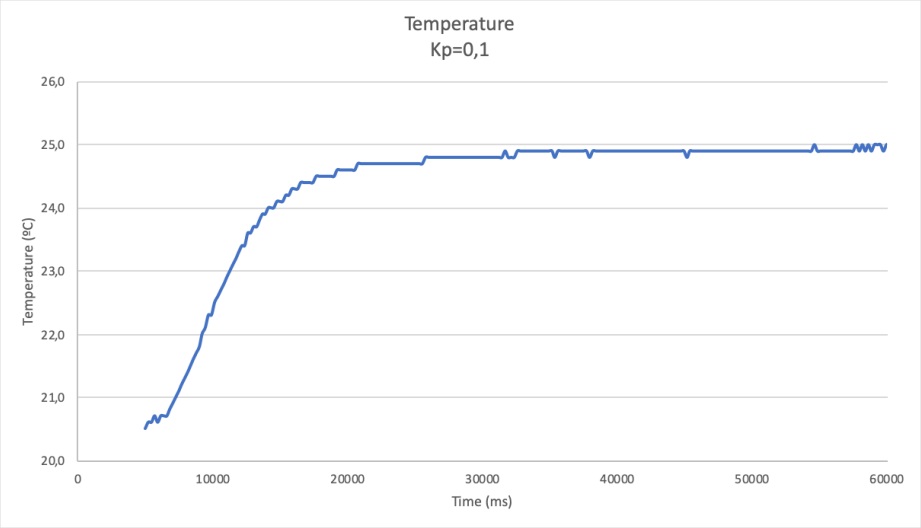
C12	1	UHE1H100MDD1TD	100µF	Nichicon	Aluminum Electrolytic Capacitors, Low Imped
C1-X	1	UPW1V101MPD1TD	100µF	Nichicon	Aluminum electrolytic capacitor 20%
C2-Y	1	UPW1V101MPD1TD	100µF	Nichicon	Aluminum electrolytic capacitor 20%
C3-Z	1	UPW1V101MPD1TD	100µF	Nichicon	Aluminum electrolytic capacitor 20%

14.3. PI CONTROLLERS FOR EACH PWM_FACTOR

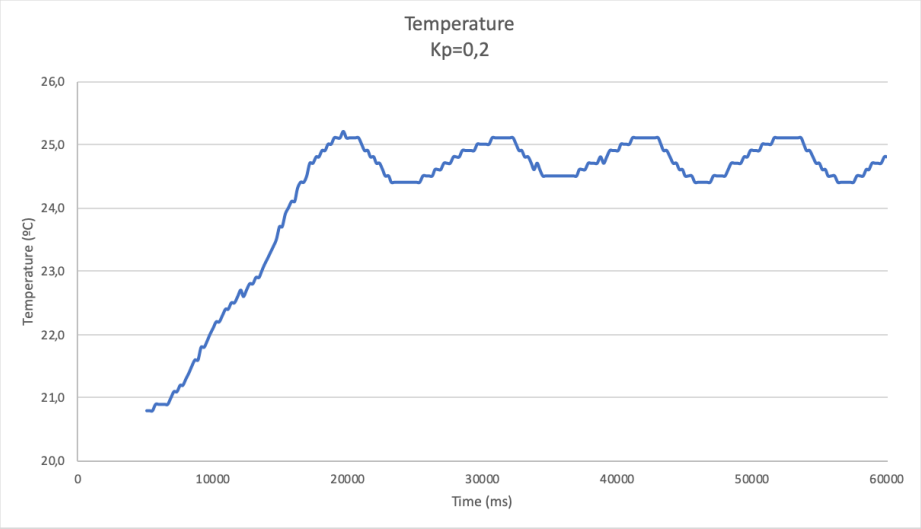
As it was explained before, for each pwm_factor a different controller was obtained when following the Ziegler-Nichols method.

For each graph, the system was first cooled to 20 ºC, once the temperature was reached and the system was steady, the set temperature was then set to 25 ºC. Each graph is a result of the different temperature measurements of the system taken every 230 milliseconds approximately. For each pwm_factor the procedure was repeated several times with an integral gain of zero and different proportional gains; until the system oscillated.

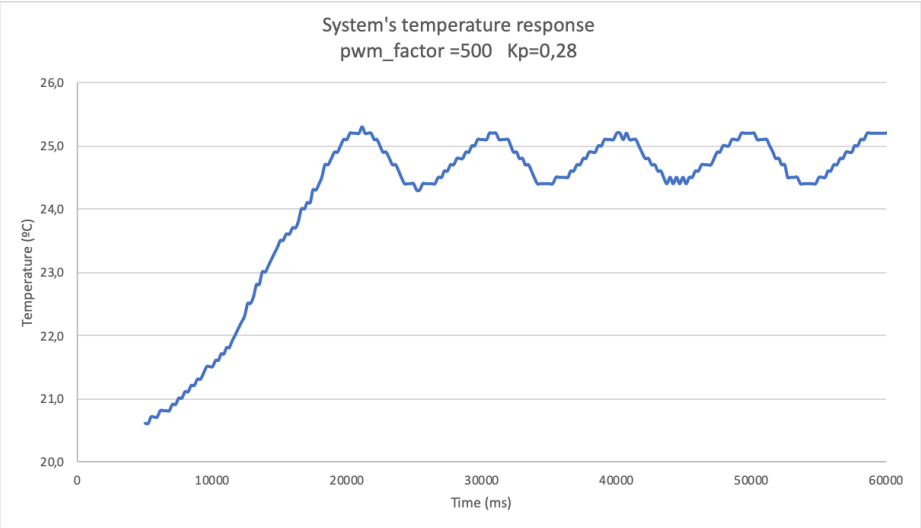
14.3.1. PWM_FACTOR OF 500



GRAPH 2 - THE 500 PWM_FACTOR-SYSTEM'S TEMPERATURE RESPONSE TO A STEP INPUT WITH A KP OF 0,1



GRAPH 3 - THE 500 PWM_FACTOR-SYSTEM'S TEMPERATURE RESPONSE TO A STEP INPUT WITH A KP OF 0,2



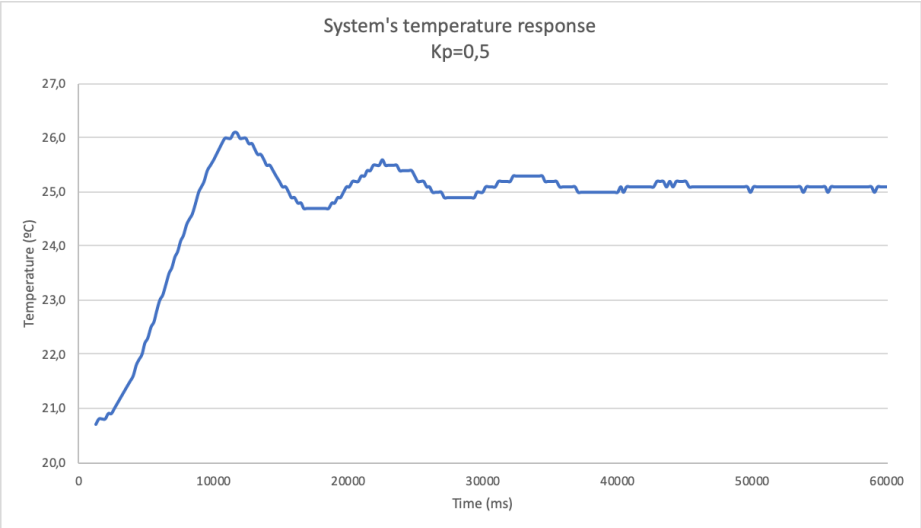
GRAPH 4 - THE 500 PWM_FACTOR-SYSTEM'S TEMPERATURE RESPONSE TO A STEP INPUT WITH A KP OF 0,28

As can be seen in the previous graphs, the system's response to a step input varies with different Kp values, as expected. On Graph 2 the response of the system does not have any oscillations, so the Kp must be incremented. On Graph 3the response of the system shows some oscillations but still not the desired ones. When tried a higher Kp, as in Graph 4, the oscillations are more consistent, and their amplitude barely varies. The final critical gain of the system is Kc = 0,28 for a pwm_factor of 500.

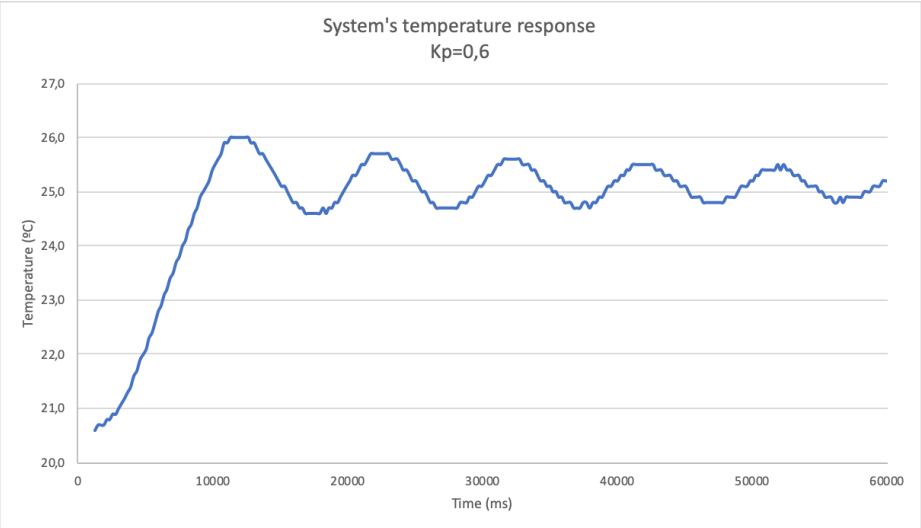
TABLE X
FINAL VALUES OF THE PWM_FACTOR=500 CONTROLLER

pwm_factor	Kc	Kp	Ki
500	0.28	0.126	0.016

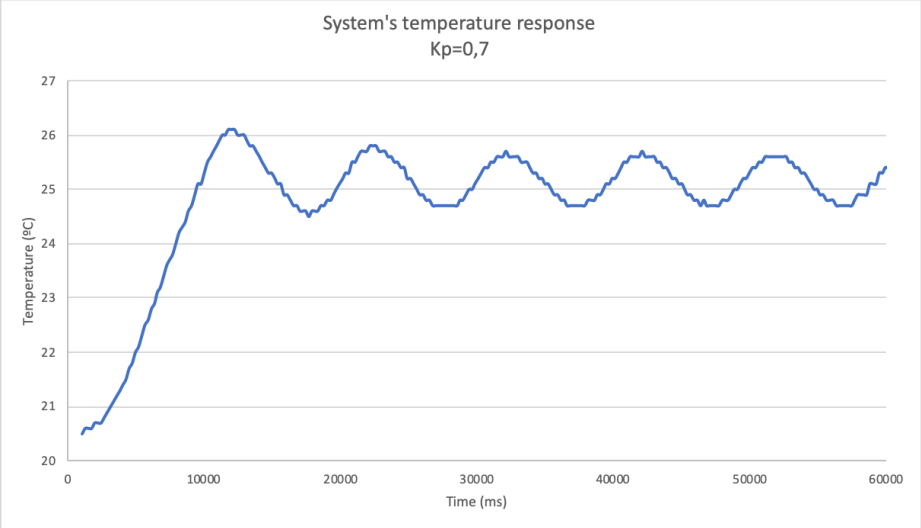
14.3.2. PWM_FACTOR OF 700



GRAPH 5 - THE 700 PWM_FACTOR-SYSTEM'S TEMPERATURE RESPONSE TO A STEP INPUT WITH A KP OF 0,5



GRAPH 6 - THE 700 PWM_FACTOR-SYSTEM'S TEMPERATURE RESPONSE TO A STEP INPUT WITH A KP OF 0,6



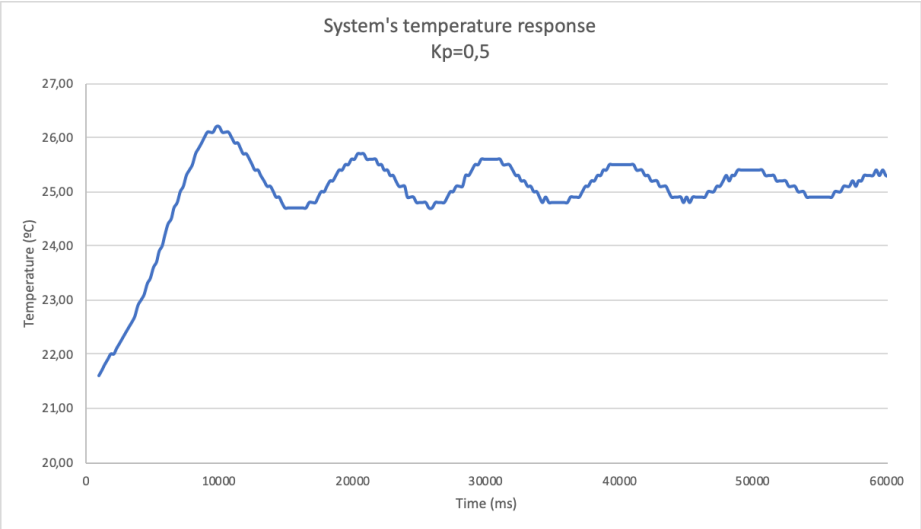
GRAPH 7 - THE 700 PWM_FACTOR-SYSTEM'S TEMPERATURE RESPONSE TO A STEP INPUT WITH A KP OF 0,7

On Graph 5 the response of the system has some oscillations at the beginning, but those fade away with time, so the Kp must be incremented. On Graph 6the response of the system shows some oscillations but those lower with time too. When tried a higher Kp, as in Graph 7, the oscillations are more consistent, and their amplitude barely varies. The final critical gain of the system is Kc = 0,7 for a pwm_factor of 700.

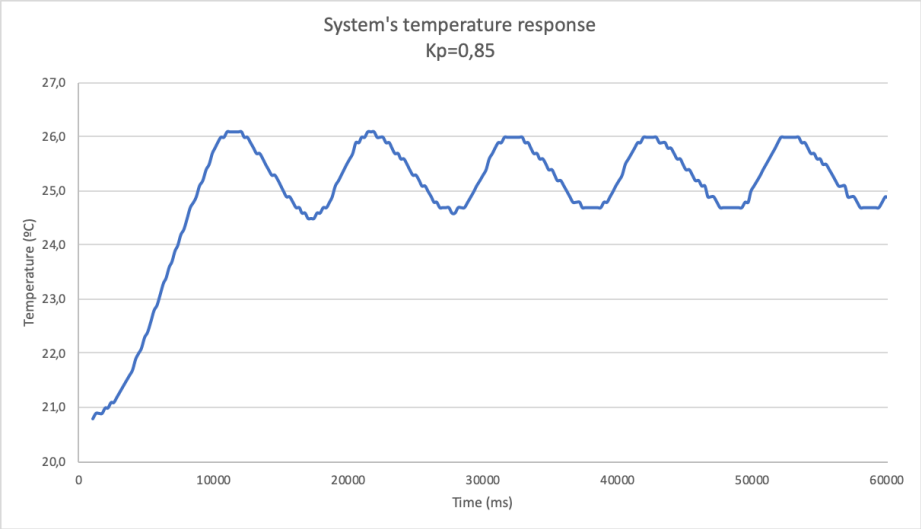
TABLE XI
FINAL VALUES OF THE PWM_FACTOR=700 CONTROLLER

pwm_factor	Kc	Kp	Ki
700	0.70	0.315	0.038

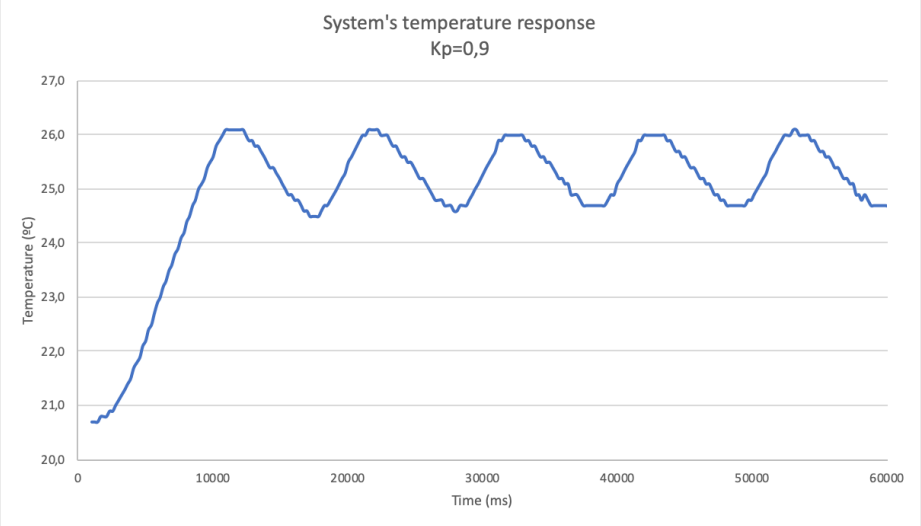
14.3.3. PWM_FACTOR OF 900



GRAPH 8 - THE 900 PWM_FACTOR-SYSTEM'S TEMPERATURE RESPONSE TO A STEP INPUT WITH A KP OF 0,5



GRAPH 9 - THE 900 PWM_FACTOR-SYSTEM'S TEMPERATURE RESPONSE TO A STEP INPUT WITH A KP OF 0,85



GRAPH 10 - THE 900 PWM_FACTOR-SYSTEM'S TEMPERATURE RESPONSE TO A STEP INPUT WITH A KP OF 0,90

On Graph 8 the response of the system has some oscillations at the beginning, but those fade away with time, so the Kp must be incremented. On Graph 9 the response of the system shows perfect oscillations that are consistent and with constant amplitude. The final Graph 10 shows another response with a greater gain, but the oscillations are really similar to those with a gain of 0,85. The final critical gain of the system is Kc = 0,85 for a pwm_factor of 900.

TABLE XII
FINAL VALUES OF THE PWM_FACTOR=900 CONTROLLER

pwm_factor	Kc	Kp	Ki
900	0.85	0.383	0.046

14.4. ARDUINO CODE FLOW CHARTS

14.4.1. MAIN CODE

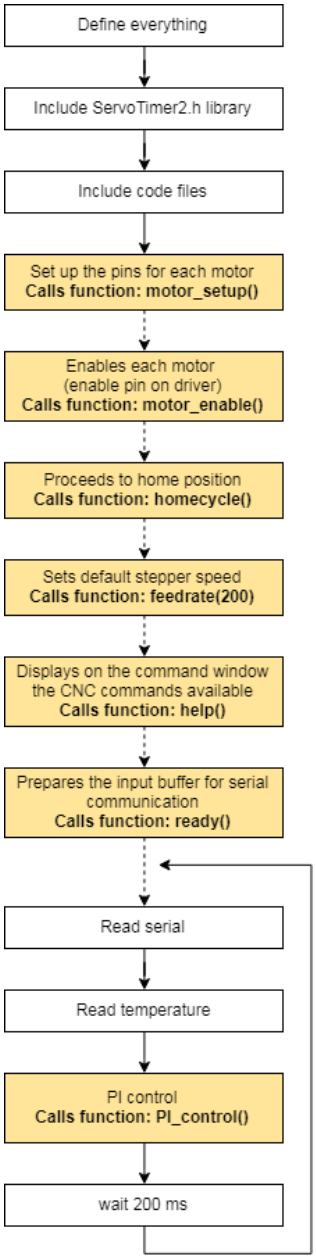


FIGURE 61 – MAIN.INO

14.4.2. COMMANDS.H

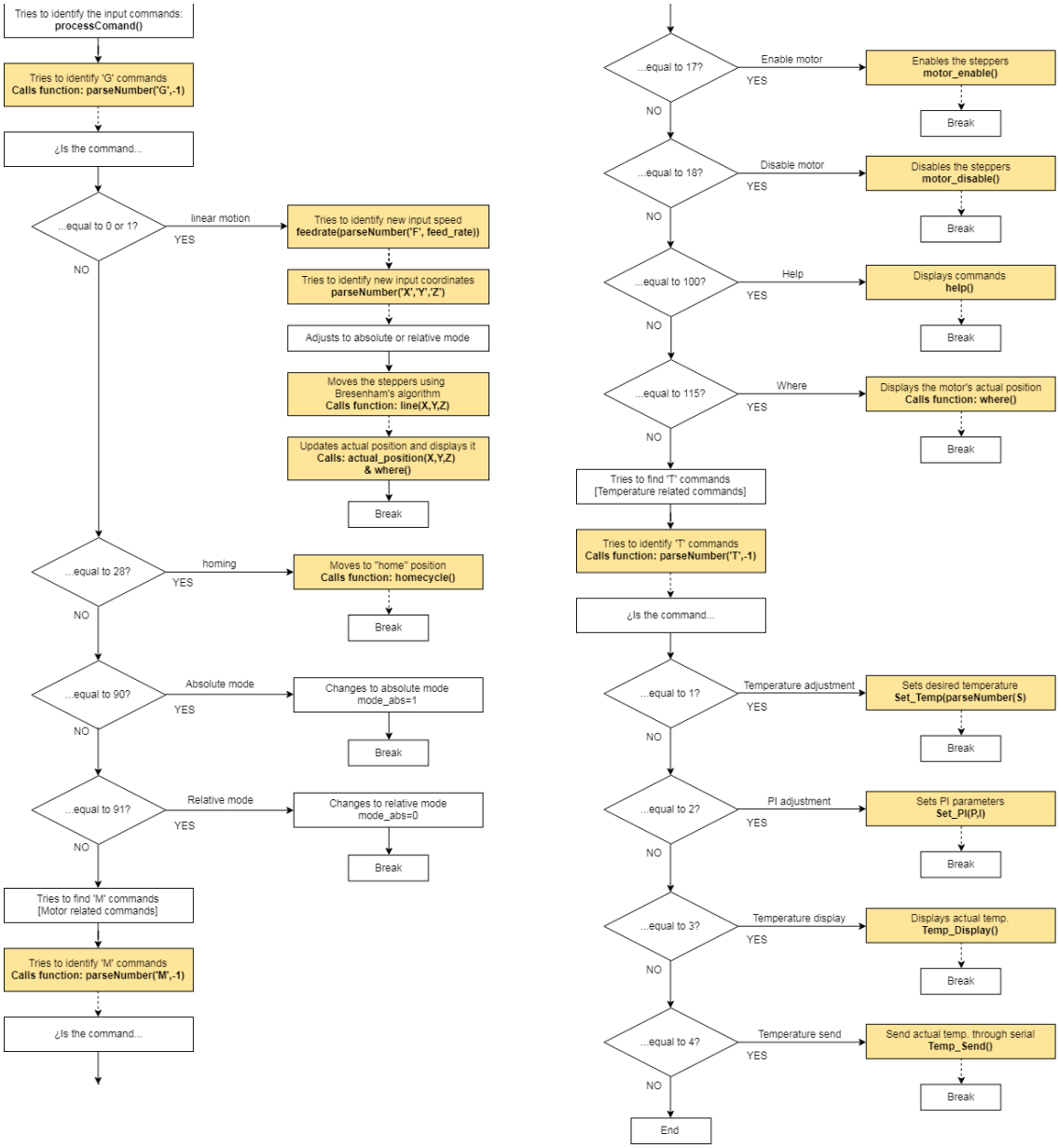


FIGURE 62 – PROCESSCOMMAND()

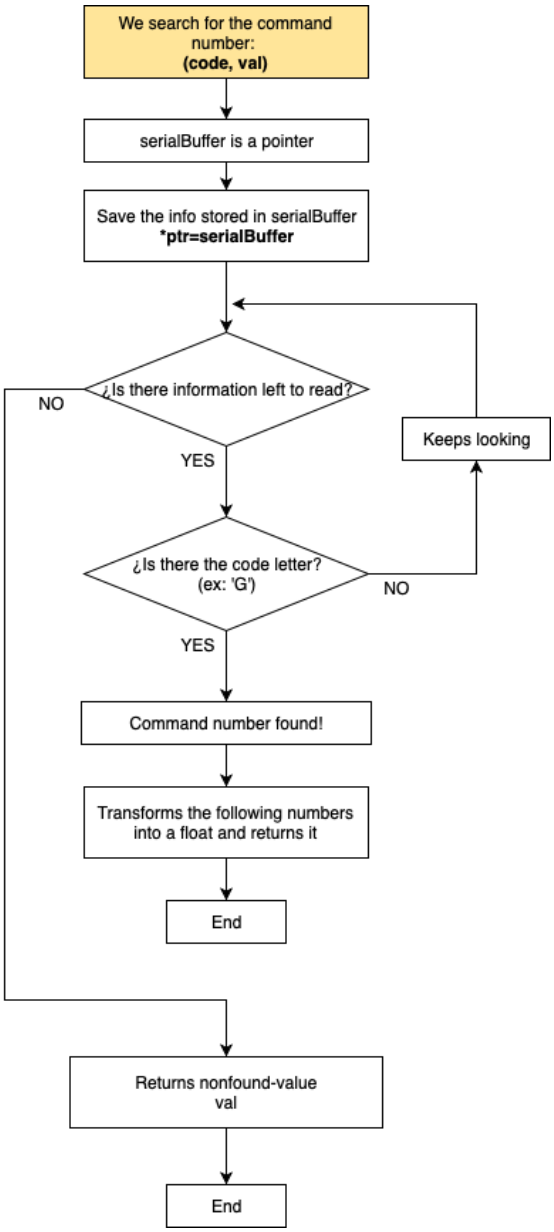


FIGURE 63 – PARSENUMBER()

14.4.4. HOMECYCLE.H

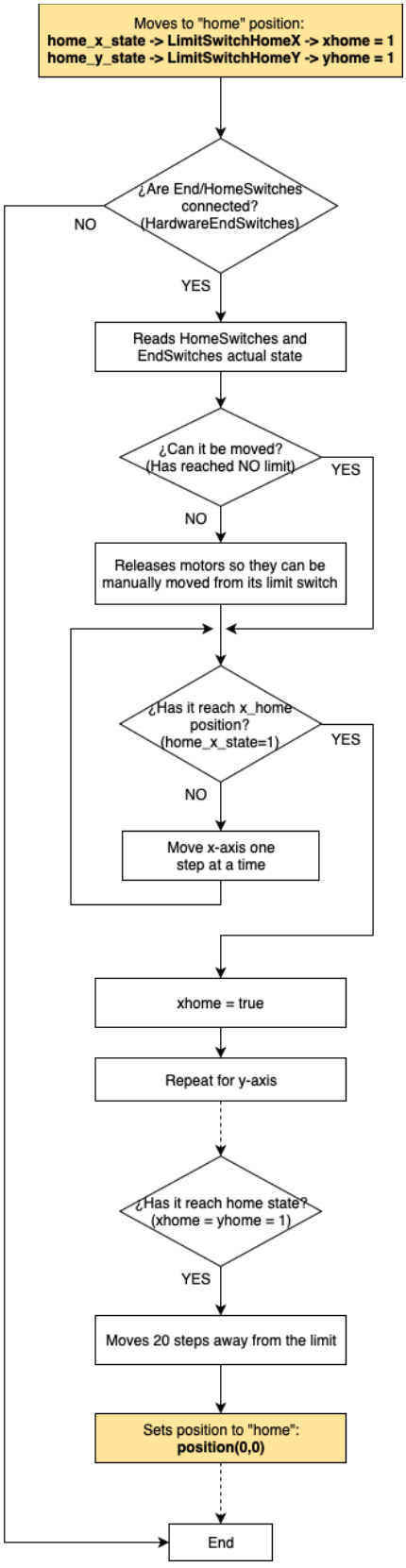


FIGURE 64 – HOMECYCLE()

14.4.5. LINE.H

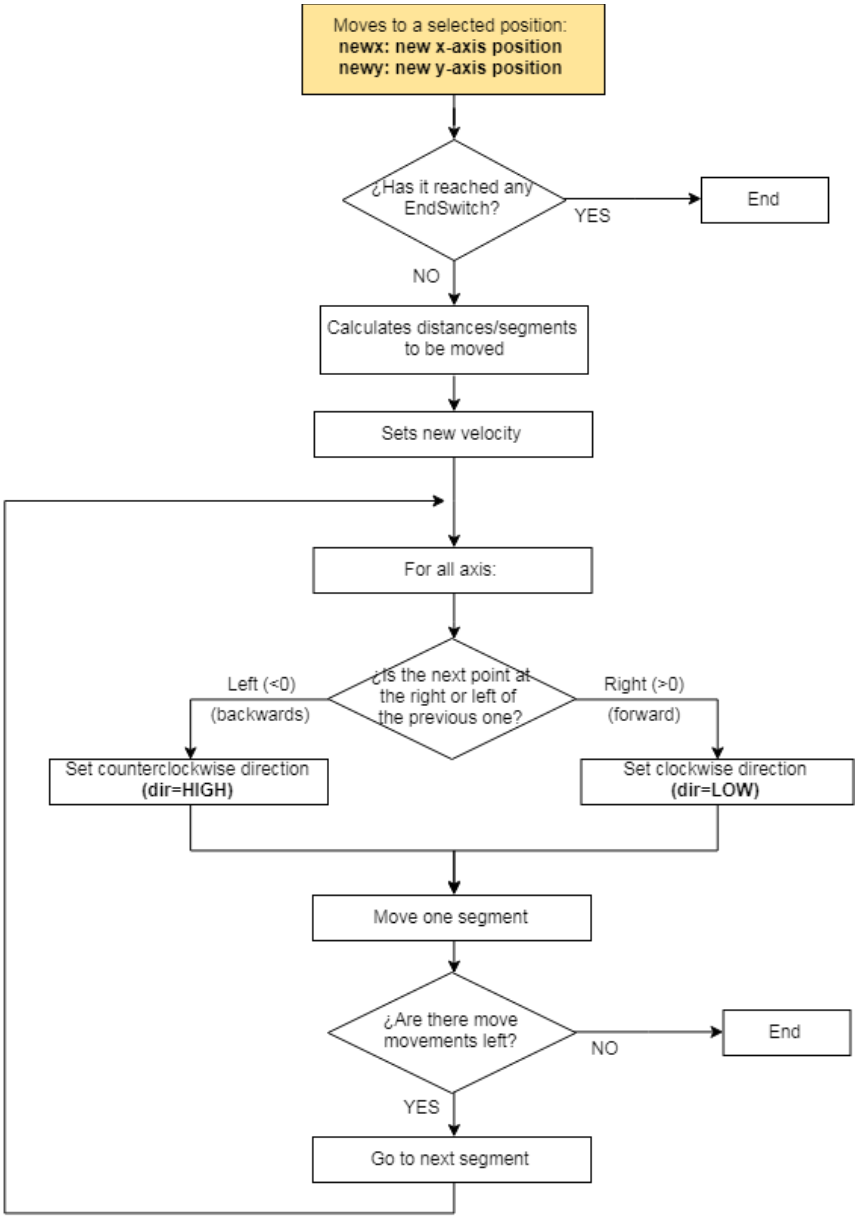


FIGURE 65 – LINE.H

14.4.6. MOTORS.H

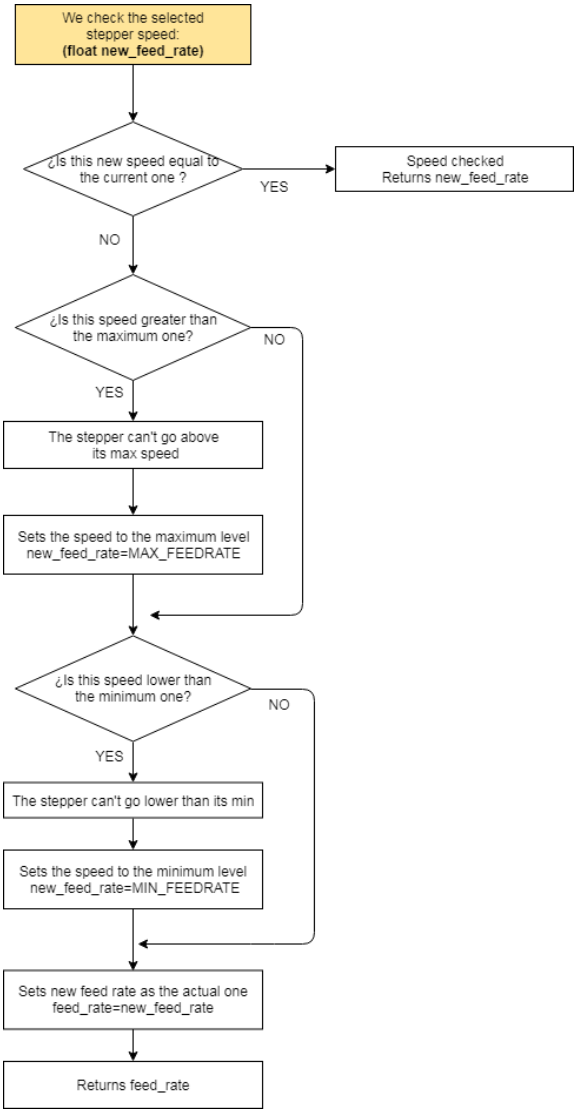


FIGURE 66 – FEEDRATE()

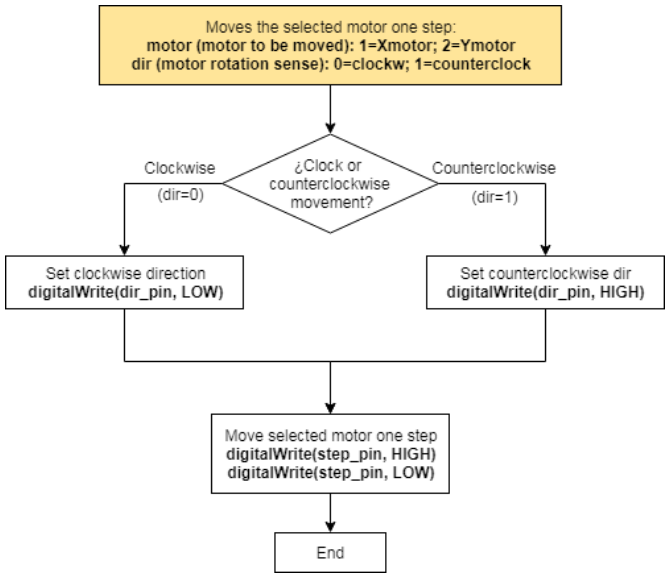


FIGURE 67 – ONESTEP()

14.4.7. PI.H

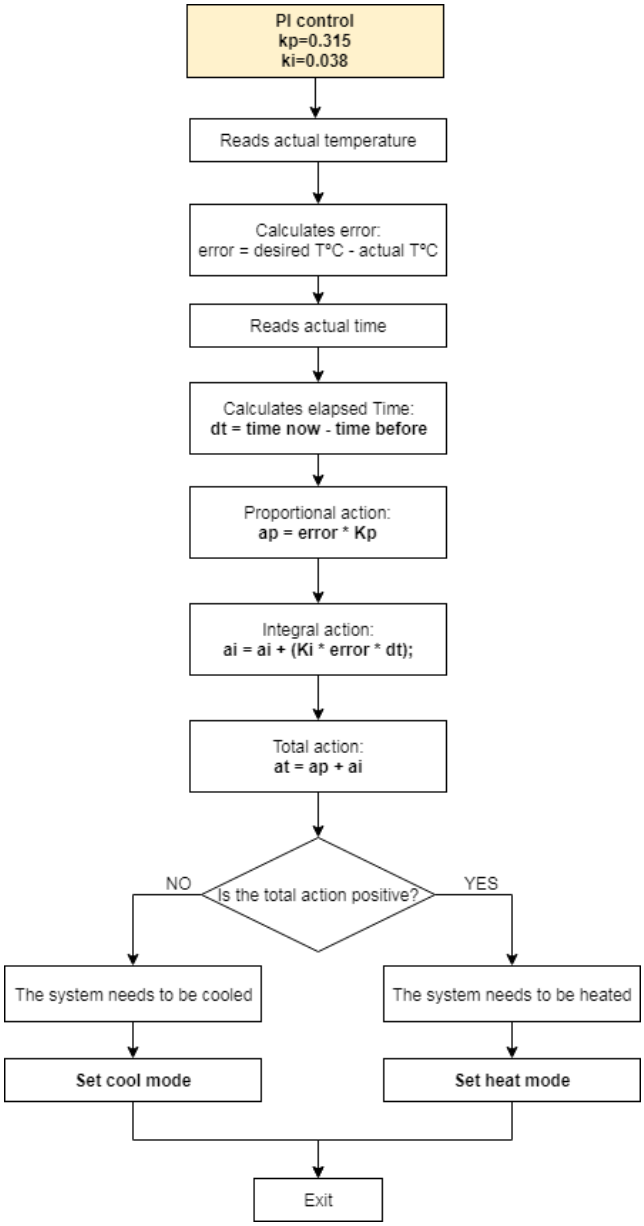


FIGURE 68 – PI_CONTROL()

14.4.8. TABLE_LOOKUP.H

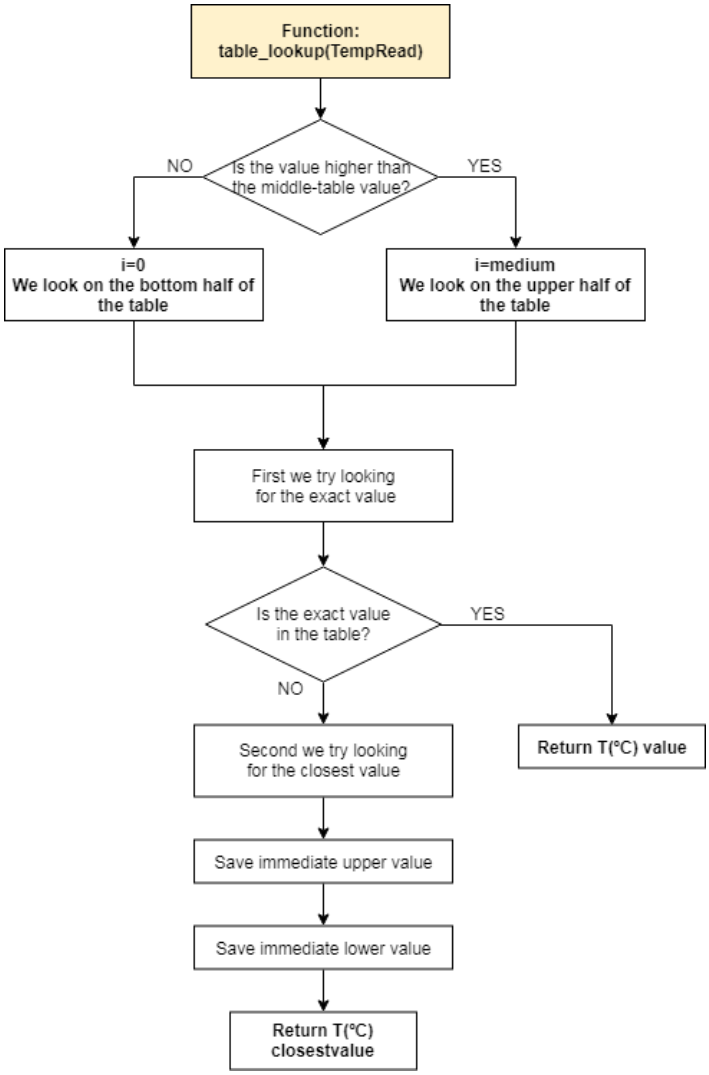


FIGURE 69 – TABLE_LOOKUP()

14.5. ARDUINO CODE

14.5.1. MAIN CODE

```
/**
 *
 * MOTORS
 * This code reads control the x,y,z motors
 * Allows for CNC input intructions/code
 *
 * THERMISTOR
 * This code reads the temperature from the thermistors
 * Also allows to use different thermistors
 * Just by creating a new thermtable and adding it to the thermtables.h file
 *
 * Steps
 * 1- Create new thermistor table "thermtable_#.h"
 * 2- Add thermistor table and number (#) to "thermtables.h"
 * 3- Add the correct thermistor number (#) to this code in "#define ThermSensor #"
 * 4- Check the rest of the settings
 *
 * Note!
 * When building the thermistor table, oversample the adc values times 16 (adc*16)
 *
 * PI CONTROL
 * This code controls the peltier cells with a PI algorithm
 *
 * NOTE:
 * A library called "ServoTimer2.h" must be downloaded and installed.
 *
 */
//**************************************************************************************************
//*DEFINITIONS
//**************************************************************************************************
//--------------------------------------------------------------------------
// OVERALL
//--------------------------------------------------------------------------

#define BAUD        (115200) //115200
#define MAX_BUF      (64)  // What is the longest message Arduino can store?

//--------------------------------------------------------------------------
// MOTORS
//--------------------------------------------------------------------------
//SETTINGS
#define MAX_FEEDRATE  (50) //50->normal, 100-> fast, 150-> super-fast; Up to 175 works
#define MIN_FEEDRATE  (1)  //1-> one step at a time

#define NUM_AXIES     (2) //XY-axis (2 or less)

#define CLOCK_FREQ    (16000000L)
#define MAX_COUNTER   (65536L)
#define MAX_SEGMENTS  (32)

#define mm_STEP     (2) //Sets 2mm = 1Step
#define MAX_DIST_mm (670) //Set maximum motor distance in mm
#define MAX_DIST_stp    (MAX_DIST_mm / mm_STEP)
bool LIMIT_DIST = 0; //Will be automatically set to 1 if the input distance is > than the
maximum
```

```
//PINS
//Motors
#define MOTOR_0_STEP_PIN          (53) //Stepper motor
#define MOTOR_0_DIR_PIN           (52)
#define MOTOR_0_ENABLE_PIN        (51)

#define MOTOR_1_STEP_PIN          (49) //Stepper motor
#define MOTOR_1_DIR_PIN           (48)
#define MOTOR_1_ENABLE_PIN        (50)

#include <ServoTimer2.h>  //Servo library that works with Timer2 instead of 1
ServoTimer2 myservo;

//EndSwitches
#define HardwareEndSwitches (1) //0=NO EndSwitches; 1=YES EndSwitches
#define LimitSwitchHomeX (40) // Pins of
#define LimitSwitchHomeY (41) // the

//VARIABLES

//STRUCTURES
// for line()
typedef struct {
  long step_count;
  long delta;  // number of steps to move
  long absdelta;
  long over;  // for dx/dy bresenham calculations
  int dir;
} Axis;

typedef struct {
  int step_pin;
  int dir_pin;
  int enable_pin;
} Motor;

typedef struct {
  Axis a[NUM_AXIES];
  int steps;
  int steps_left;
  long feed_rate;
} Segment;

//GLOBALS
Axis a[NUM_AXIES];  // for line()
Axis atemp;  // for line()
Motor motors[NUM_AXIES];

Segment line_segments[MAX_SEGMENTS];
volatile int current_segment;
volatile int last_segment;
unsigned int step_loops;

char serialBuffer[MAX_BUF];  // where we store the message until we get a ';'
int sofar;  // how much is in the buffer

//homing
boolean has_origin = false; //helps us to know when the origin has been reached
```

```
// speeds
float feed_rate=0;  // human version
float px,py,pz,pu,pv,pw;  // actual position

// settings
char mode_abs=1;  // absolute mode?
long line_number=0;

//------------------------------------------------------------------------------
// THERMISTOR – TEMPERATURE CONTROL
//------------------------------------------------------------------------------
//THERMISTOR
//Define
#define ThermSensor (22) //Your thermistor number
#define ThermPin (A0)    //The thermistor pin (Analog)

//Code
#include "thermtables.h" //Include the thermistors tables

//Variables
int TempRead = 0;
float Temp = 0;
int rows=sizeof (thermtable) / sizeof (thermtable[0]); //calculates the rows-dimention of the
table
int maxTemp = thermtable[rows-1][1];
int maxTempbits = thermtable[rows-1][0];
int minTemp = thermtable[0][1];
int minTempbits = thermtable[0][0];
bool Temp_error=0; //Will be set to 1 if there's an error

//PELTIER
//Define Peltier pins
#define pwm (13) //Cantidad de corriente (Rápido/Lento)
#define IN1 (12)  //Dirección de la corriente
#define IN2 (11)  //(IN1=HIGH -> Calentar/ IN2=HIGH -> Enfriar)

//Variables
#define pwm_factor (700) //Adjustable factor for the PWM output
int pwm_value = 0; //Cantidad de corriente (Rápido/Lento)
byte state = 0; // 0:OFF y 1:Enfriar y 2:Calentar (Previous state)
float Desired_Temp = 25; //We start with a desired temperature of 26ºC

//LIBRARIES
//Overall
#include "Functions.h"
#include "Setup.h"
#include "Commands.h"
//Motors
#include "line.h"
#include "timer.h"
#include "homecycle.h"
#include "motors.h"
//Thermistor
#include "table_lookup.h" //Include the table search algorithm
#include "PI.h" //Include the PI control code

//MORE INFO
#define INFO     (1)
#ifdef INFO
```

```
  #include "Info.h" //Include to know more about the temperature control
 #endif
 //*****************************************************************************************
 //*SETUP
 //*****************************************************************************************
 /**
  * First thing this machine does on startup.  Runs only once.
  */
 void setup() {
  Serial.begin(BAUD);  // open coms
  //MOTORS
  motor_setup();
  motor_enable();
  Serial.println("Wait 'homing' in process...");  // repeat it back so I know you got the message
  if (HardwareEndSwitches) {
    pinMode(LimitSwitchHomeX, INPUT_PULLUP);
    pinMode(LimitSwitchHomeY, INPUT_PULLUP);
  }
  homecycle(); //Performs homing at the beginning
  eppendorf(10);
  eppendorf(0);
  actual_position(0,0,0);  // set staring position
  feedrate(200);  // set default speed

  //TEMPERATURE CONTROL
  //Pin declaration
  pinMode (ThermPin, INPUT);
  pinMode (IN1, OUTPUT);
  pinMode (IN2, OUTPUT);

  #ifdef INFO
    motor_info();
    therm_info();
    PI_info();
    delay(3000);
  #endif

  Serial.print("Everything is ready!\n");
  help();  // say hello
  ready();
  delay(500);
 }

 //*****************************************************************************************
 //*MAIN
 //*****************************************************************************************
 /**
  * After setup() this machine will repeat loop() forever.
  */
 void loop() {
  // listen for serial commands
  while(Serial.available() > 0) {  // if something is available
    char c=Serial.read();  // get it
    Serial.print(c);  // repeat it back so I know you got the message
    if(sofar<MAX_BUF-1) serialBuffer[sofar++]=c;  // store it
    if(c=='\n') {
      // entire message received
      serialBuffer[sofar]=0;  // end the buffer so string functions work right
      Serial.print(F("\r\n"));  // echo a return character for humans
```

```
    processCommand();  // do something with the command
    ready();
   }
  }

  //TEMPERATURE READ
  TempRead = 16*analogRead(ThermPin); //read the input temperature from pin A0
  if (TempRead>maxTempbits && TempRead<minTempbits){ //First we check if the value is
within limits
   Temp_error=0;                          //remember that bits are inversely proportional to Tª
   Temp=table_lookup(TempRead, rows); //searches in the table
   Temp=Temp/10; //searches in the table
  } else if (Temp_error==0){ //error detected
   Temp_error=1;
   Serial.print("\nTEMPERATURE SENSOR ERROR");
  }

  //PELTIER CONTROL- PI
  if( abs(Temp-Desired_Temp)<=2 && aux_PI==0){
   Ki_out=1;
   aux_PI=1; //We put it back to zero every time the Desired_Temp changes.
  }
  at = PI_control(Temp); //PI control function
  pwm_value = (abs(at)*pwm_factor);
  if (pwm_value>255){
   pwm_value=255;
  }
  analogWrite(pwm, pwm_value);

  delay(200); //delay in between reads for stability
}
```

14.5.2. COMMANDS

```
/**
 * This library includes the functions that deal with input commands.
 *
 * The following functions are included here:
 *    void processCommand();
 *    float parseNumber(char code,float val);
 *    void help();
 *    void where();
 *    void output(char *code,float val);
 *
 */
/**
 * Read the input buffer and find any recognized commands.  One G or M command per line.
 */

void processCommand() {
  int cmd = parseNumber('G',-1);
  switch(cmd) {
  case  0:
  case  1: { // line
     Max_Dist (mm_step(parseNumber('X',(mode_abs?px:0)) + (mode_abs?0:px)),
            mm_step(parseNumber('Y',(mode_abs?py:0)) + (mode_abs?0:py)));
     if (LIMIT_DIST){
       LIMIT_DIST=0;
     } else {
```

```
      feedrate(parseNumber('F',feed_rate));
      line( mm_step(parseNumber('X',(mode_abs?px:0)) + (mode_abs?0:px)),
          mm_step(parseNumber('Y',(mode_abs?py:0)) + (mode_abs?0:py)),
          mm_step(parseNumber('Z',(mode_abs?pz:0)) + (mode_abs?0:pz)),
          mm_step(parseNumber('U',(mode_abs?pu:0)) + (mode_abs?0:pu)),
          mm_step(parseNumber('V',(mode_abs?pv:0)) + (mode_abs?0:pv)),
          mm_step(parseNumber('W',(mode_abs?pw:0)) + (mode_abs?0:pw)),
          feed_rate );
      eppendorf(parseNumber('Z',(mode_abs?pz:0)) + (mode_abs?0:pz)); //Moves Z-axis
      actual_position( parseNumber('X',(mode_abs?px:0)) + (mode_abs?0:px), //Update actual
position
          parseNumber('Y',(mode_abs?py:0)) + (mode_abs?0:py),
          parseNumber('Z',(mode_abs?pz:0)) + (mode_abs?0:pz));
     }
    where(); //Prints actual position
    break;
   }
 case 28:  homecycle(); break;  //homing
 case 90:  mode_abs=1; Serial.println("Absolute mode selected"); break;  // absolute mode
 case 91:  mode_abs=0; Serial.println("Relative mode selected"); break;  // relative mode
 default:  break;
 }

 cmd = parseNumber('M',-1);
 switch(cmd) {
 case 17:  motor_enable();  break;
 case 18:  motor_disable();  break;
 case 100:  help();  break;
 case 115:  where();  break; //Prints actual position + speedrate + mode
 default:  break;
 }

 cmd = parseNumber('T',-1);
 switch(cmd) {
 case 1:  Set_Temp(parseNumber('S',25)); break; //Sets 25ºC as default
 case 2:  Set_PI(parseNumber('P',999),
             parseNumber('I',999)); break;
 case 3:  Temp_Display(); break;
 case 4:  Temp_Send(); break; //Sends actual temperature
 default:  break;
 }
}

/**
 * Look for character /code/ in the buffer and read the float that immediately follows it.
 * @return the value found.  If nothing is found, /val/ is returned.
 * @input code the character to look for.
 * @input val the return value if /code/ is not found.
 **/
float parseNumber(char code,float val) {
 char *ptr=serialBuffer;  // start at the beginning of buffer
 while((long)ptr > 1 && (*ptr) && (long)ptr < (long)serialBuffer+sofar) {  // walk to the end
  if(*ptr==code) {  // if you find code on your walk,
   return atof(ptr+1);  // convert the digits that follow into a float and return it
  }
  ptr=strchr(ptr,' ')+1;  // take a step from here to the letter after the next space
 }
 return val;  // end reached, nothing found, return default val.
}
```

```
/**
 * display helpful information
 */
void help() {
  Serial.print(F("GcodeCNCDemo6AxisV2 "));
  Serial.println(F("Commands:"));
  Serial.println(F("G00/G01 [X/Y(steps)] [Z(degrees)] [F(feedrate)]; - linear move"));
  Serial.println(F("G04 P[seconds]; - delay"));
  Serial.println(F("G28; - homing"));
  Serial.println(F("G90; - absolute mode"));
  Serial.println(F("G91; - relative mode"));
  Serial.println(F("M17; - enable motors"));
  Serial.println(F("M18; - disable motors"));
  Serial.println(F("M100; - this help message"));
  Serial.println(F("M115; - report actual position and feedrate"));
  Serial.println(F("T1 [S(temperature)]; - Sets desired temperature"));
  Serial.println(F("T2 [P(Kp value)] [I(Ki value)]; - Sets PI values"));
  Serial.println(F("T3  - Displays actual temperature and desired temperature"));
  Serial.println(F("T4  - Sends actual temperature"));
  Serial.println(F("All commands must end with a newline."));
}

/**
 * write a string followed by a float to the serial line.  Convenient for debugging.
 * @input code the string.
 * @input val the float.
 */
void output(char *code,float val) {
  Serial.print(code);
  Serial.println(val);
}

/**
 * print the current position, feedrate, and absolute mode.
 */
void where() {
  Serial.println("The actual motor position with respect to its home-position is: ");
  output("X", px);
  output("Y", py);
  output("Z", pz);
  output("F",feed_rate);
  Serial.println(mode_abs?"ABS":"REL");
}

/**
 * sends current temperature to display on screen.
 */
void Temp_Send() {
  Serial.println(Temp);
}
```

14.5.3. FUNCTIONS

```
/**
 * Functions used
 */
```

```
//Commands.h
void processCommand();
float parseNumber(char code,float val);
void help();
void where();
void output(char *code,float val);
void Temp_Send();

//homecycle.h
void homecycle();

//line.h
void line(int n0,int n1,int n2,int n3,int n4,int n5,float new_feed_rate);
int get_next_segment(int i);
int get_prev_segment(int i);

//timer_set_frequency.h
void timer_set_frequency(long desired_freq_hz);

//motors.h
void eppendorf(int deg);
void onestep(int motor, int dir);
float feedrate(float new_feed_rate);
void motor_enable();
void motor_disable();
void actual_position(float npx,float npy,float npz);

//PI.h
void Temp_Display();
void Set_Temp(float Temp);
void Set_PI(float P, float I);
void cool();
void heat();
float PI_control(float Temp);

//Info.h
void motor_info();
void therm_info();
void PI_info();
```

14.5.4. INFO

```
/**
 * This info will be displayed at the beginning of the code if wanted
 *
 */

//FUNCTIONS
void motor_info(){
  Serial.println("\n~~MOTOR INFO~~\n");
  Serial.print("Number of working motors: ");
  Serial.println(NUM_AXIES);
  Serial.print("Maximum velocity: ");
  Serial.println(MAX_FEEDRATE);
  Serial.print("Minimum velocity: ");
  Serial.println(MIN_FEEDRATE);

  Serial.print("\n");
```

```
}

void therm_info(){
 Serial.println("\n~~THERMISTOR INFO~~\n");
 //MAX temp reading
 Serial.print("Maximum readable temperature (ºC): ");
 Serial.println(maxTemp/10,1);
 //MIN temp reading
 Serial.print("Minimum readable temperature (ºC): ");
 Serial.println(minTemp/10,1);
 Serial.print("\n");
}

void PI_info(){
 Serial.println("\n~~PI-CONTROL INFO~~\n");
 //Kp value
 Serial.print("Kp value: ");
 Serial.println(Kp);
 //Ki value
 Serial.print("Ki value: ");
 Serial.println(Ki);
 Serial.print("\n");
}
```

14.5.5. PI

```
/**
 * PI control code
 *
 * FUNCTIONS:
 *    void Temp_Display();
 *    float PI_control(float Temp);
 *    void Set_Temp(float Temp);
 *    void Set_PI(float P, float I);
 */
//PI CONSTANTS
float Kp = 0.315;
float Ki = 0.038;

//VARIABLES
float ap = 0; //proportional action
float ai = 0; //integral action
float at = 0; //total action
float error = 0; //error
int Time_now = 0;  //Actual time
int Time_prev = 0; //Previous time
float dt = 0; //Time difference
bool Ki_out=0;
bool aux_PI = 0;

//FUNCTIONS
/**
 * Function that displays the Desired temperature and the Actual Temp. as well
 */
void Temp_Display(){
 Serial.println("The actual temperature of the cell is: ");
 Serial.print(Temp,1);
 Serial.print(" ºC\n");
```

```
  Serial.println("The desired temperature of the cell is: ");
  Serial.print(Desired_Temp,1);
  Serial.print(" ºC\n");
}

/**
 * Function that sets the Desired temperature
 */
void Set_Temp(float Temp){
  if (Temp!=999){
    Desired_Temp=Temp;
    aux_PI=0; //We reset it
  }
  Serial.println("Desired temperature set to: ");
  Serial.print(Desired_Temp,1);
  Serial.print(" ºC\n");
}

/**
 * Function that sets the PI parameters
 */
void Set_PI(float P, float I){
  if (P!=999){
    Kp = P;
  }
  if (I!=999){
    Ki = I;
  }
  PI_info();
}

/**
 * Function that has the PI control algorithm
 */
float PI_control(float Temp){
  Time_now = millis(); //Reads actual time
  error = Desired_Temp - Temp; //Calculates error
  //Proportional
  ap = error * Kp;
  //Integral
  dt = (Time_now - Time_prev)/1000;
  ai = ai + (Ki * error * dt * Ki_out);
  //Total
  Time_prev = Time_now;
  at = ap + ai;

  if (at>0 && state!=2){ //The system needs to be heated
    heat();
    state=2;
  } else if (at<0 && state!=1){ //The system needs to cool down
    cool();
    state=1;
  }
  return at;
}

/**
 * Function that sets the cooling mode
 */
```

```
void cool(){
  digitalWrite (IN2, LOW);
  digitalWrite (IN1, HIGH);
}

/**
 * Function that sets the heating mode
 */
void heat(){
  digitalWrite (IN1, LOW);
  digitalWrite (IN2, HIGH);
}
```

14.5.6. SETUP

```
/**
 * This library includes the functions that deal with the setup of the system.
 *
 * The following functions are included here:
 *    void Max_Dist(int X,int Y,int Z)
 *    int mm_step(int stps);
 *    int step_mm(int mm);
 *    void ready();
 *    void motor_setup();
 *
 */
/**
 * prepares the input buffer to receive a new message and tells the serial connected device it is
ready for more.
 */
void ready() {
  sofar=0;  // clear input buffer
  Serial.print(F(">"));  // signal ready to receive input
}

/**
 * set up the pins for each motor
 */
void motor_setup() {
  motors[0].step_pin = MOTOR_0_STEP_PIN;
  motors[0].dir_pin = MOTOR_0_DIR_PIN;
  motors[0].enable_pin = MOTOR_0_ENABLE_PIN;

  motors[1].step_pin = MOTOR_1_STEP_PIN;
  motors[1].dir_pin = MOTOR_1_DIR_PIN;
  motors[1].enable_pin = MOTOR_1_ENABLE_PIN;

  int i;
  for(i=0;i<NUM_AXIES;++i) {
    // set the motor pin & scale
    pinMode(motors[i].step_pin,OUTPUT);
    pinMode(motors[i].dir_pin,OUTPUT);
    pinMode(motors[i].enable_pin,OUTPUT);
  }

  myservo.attach(10); //Servo motor

}
```

```
/**
 * Function that check the maximum distance
 */
void Max_Dist(int X,int Y){
  if (X>=MAX_DIST_stp) {
    LIMIT_DIST=1;
    Serial.print("Exceeds maximum X-distance\n");
  }
  if (Y>=MAX_DIST_stp) {
    LIMIT_DIST=1;
    Serial.print("Exceeds maximum Y-distance\n");
  }
  if (LIMIT_DIST){
    Serial.print("Non achievable X-Y distance, please enter new distance\n");
  }
}

/**
 * Function that transforms the number of mm in steps
 */
int mm_step(int stps){ //mm_STEP = Xmm/1step
  int mm=0;
  mm=stps/mm_STEP;
  return mm;
}

/**
 * Function that transforms the number of steps in mm
 */
int step_mm(int mm){
  int steps=0;
  steps=mm*mm_STEP;
  return steps;
}
```

14.5.7. HOMECYCLE

```
/**
 * This library includes the function that deals with the homing of the xy_system.
 *
 * The following function is included here:
 *    void homecycle();
 *
 */
/**
 *
 * Moves the motors to their home position.
 *
 */

void homecycle() {
  if (HardwareEndSwitches) {
    // get states of limit switches
    int home_x_state = digitalRead(LimitSwitchHomeX);
    int home_y_state = digitalRead(LimitSwitchHomeY);
    boolean xhome = false; // set to true when X has reached home position
    boolean yhome = false; // set to true when Y has reached home position
```

```
  //check if X & Y are on a limit and cannot move
  if (home_x_state == 0 || home_y_state == 0) {
   Serial.println("Either X or Y are not in a moveable position!");
   motor_disable(); //realeases motors so they can be moved manually
   delay(250);
   Serial.println("Motors are released. Now please move X and/or Y from its limit switch.");
   return;
  }
  while(home_x_state == 1) { // while it hasn't reach the end...
   onestep(0,1); // move x-motor[0] 1 step
   delay(250);
   home_x_state = digitalRead(LimitSwitchHomeX); // read limit switch status
   if (home_x_state == 0) { //when the limit switch has been reached:
    xhome = true; //set home variable to TRUE
    Serial.println("\nreached Home X");
    delay(250);
   }
  }
  while(home_y_state == 1) {
   onestep(1,1); // move y-motor[1] 1step
   delay(250);
   home_y_state = digitalRead(LimitSwitchHomeY); // query status of limit switch
   if (home_y_state == 0) {
    yhome = true;
    Serial.println("reached Home Y");
    delay(250);
   }
  }
  if ( xhome == true && yhome == true) {
   for (int i = 0; i < 21; i++) {  //moves 20 steps away from limit switch .....Adjust
this to our needs
    onestep(0,0); //x-motor [0], direction clockwise
    onestep(1,0); //y-motor [1]
    delay(50); // not so fast ...
   }
   actual_position(0,0,0);
   has_origin = true;
   Serial.println("Set X/Y to 0/0 ...");
   where(); //incluir función que manda la posición obtenida a la pantalla
  }
 }
}
```

14.5.8. LINE

```
/**
 * This library includes the functions that deal with the line movement of the xy_motors.
 *
 * The following functions are included here:
 *    void line(int n0,int n1,int n2,int n3,int n4,int n5,float new_feed_rate);
 *    int get_next_segment(int i);
 *    int get_prev_segment(int i);
 */

/**
 * Uses bresenham's line algorithm to move both motors
 **/
```

```
void line(int n0,int n1,int n2,int n3,int n4,int n5,float new_feed_rate) {
  int next_segment = get_next_segment(last_segment);
  while( next_segment == current_segment ) {
    // the buffer is full, we are way ahead of the motion system
    delay(1);
  }
  Segment &new_seg = line_segments[last_segment];
  new_seg.a[0].step_count = n0; // New position (nº of steps)
  new_seg.a[1].step_count = n1;
  new_seg.a[2].step_count = n2;
  new_seg.a[3].step_count = n3;
  new_seg.a[4].step_count = n4;
  new_seg.a[5].step_count = n5;

  int i;

  Segment &old_seg = line_segments[get_prev_segment(last_segment)];
  new_seg.a[0].delta = n0 - old_seg.a[0].step_count; // Calculates the number of steps it needs
to do with respect to the previous movement
  new_seg.a[1].delta = n1 - old_seg.a[1].step_count;
  new_seg.a[2].delta = n2 - old_seg.a[2].step_count;
  new_seg.a[3].delta = n3 - old_seg.a[3].step_count;
  new_seg.a[4].delta = n4 - old_seg.a[4].step_count;
  new_seg.a[5].delta = n5 - old_seg.a[5].step_count;

  new_seg.steps=0;
  new_seg.feed_rate=new_feed_rate;

  for(i=0;i<NUM_AXIES;++i) {
    new_seg.a[i].over = 0;
    new_seg.a[i].dir = (new_seg.a[i].delta > 0 ? LOW:HIGH);
    new_seg.a[i].absdelta = abs(new_seg.a[i].delta);
    if( new_seg.steps < new_seg.a[i].absdelta ) {
      new_seg.steps = new_seg.a[i].absdelta;
    }
  }

  if( new_seg.steps==0 ) return;

  new_seg.steps_left = new_seg.steps;

  if( current_segment==last_segment ) {
    timer_set_frequency(new_feed_rate);
  }
  last_segment = next_segment;
}

int get_next_segment(int i) {
  return ( i + 1 ) % MAX_SEGMENTS;
}

int get_prev_segment(int i) {
  return ( i + MAX_SEGMENTS - 1 ) % MAX_SEGMENTS;
}
```

14.5.9. MOTORS

```
/**
```

```
 * This library includes the functions that deal with the movement of the motors.
 *
 * The following functions are included here:
 *    void eppendorf(int deg)
 *    void onestep(int motor, int dir);
 *    float feedrate(float new_feed_rate);
 *    void motor_enable();
 *    void motor_disable();
 *    void actual_position(float npx,float npy,float npz,float npu,float npv,float npw);
 *
 */

void eppendorf(int deg) {
 int i=0;
 int val=0;
 if (deg>160) { //The servo can move 160 as maximum
   deg=160;
 }
 if (deg>pz){
   for (i = pz; i<= deg; ++i){
     val = map(i, 0, 160, 750, 2250);
     myservo.write(val);
     delay(15);
   }
 }
 if (deg<pz){
   for (i = pz; i>= deg; --i){
     val = map(i, 0, 160, 750, 2250);
     myservo.write(val);
     delay(15);
   }
 }
}

/**
 * Supports movement with both styles of Motor Shield
 * @input newx the destination x position
 * @input newy the destination y position
 **/
void onestep(int motor, int dir) {

 if (dir==1){ //We set the direction
   digitalWrite(motors[motor].dir_pin, HIGH);
 } else if (dir==0){
   digitalWrite(motors[motor].dir_pin, LOW);
 }
 digitalWrite(motors[motor].step_pin,HIGH);
 digitalWrite(motors[motor].step_pin,LOW);
}

/**
 * Set the feedrate (speed motors will move)
 * @input new_feed_rate the new speed in steps/second
 */
float feedrate(float new_feed_rate) {
 if(feed_rate==new_feed_rate) return new_feed_rate;  // same as last time?  quit now.

 if(new_feed_rate>MAX_FEEDRATE) {
   Serial.print(F("Feedrate set to maximum ("));
```

```
    Serial.print(MAX_FEEDRATE);
    Serial.println(F("steps/s"));
    new_feed_rate=MAX_FEEDRATE;
   }
   if(new_feed_rate<MIN_FEEDRATE) {  // don't allow crazy feed rates
    Serial.print(F("Feedrate set to minimum ("));
    Serial.print(MIN_FEEDRATE);
    Serial.println(F("steps/s"));
    new_feed_rate=MIN_FEEDRATE;
   }
   feed_rate=new_feed_rate;

   return feed_rate;
}

/**
 * Enables the motors to do NOT be able to manipulate them manually
 */
void motor_enable() {
  int i;
  for(i=0;i<NUM_AXIES;++i) {
    digitalWrite(motors[i].enable_pin,LOW); //the enable pin is denied
  }
}

/**
 * Disables the motors to be able to manipulate them manually
 */
void motor_disable() {
  int i;
  for(i=0;i<NUM_AXIES;++i) {
    digitalWrite(motors[i].enable_pin,HIGH); //the enable pin is denied
  }
}

/**
 * Saves the current position..
 */
void actual_position(float npx,float npy,float npz) {
  px=npx;
  py=npy;
  pz=npz;
}
```

14.5.10. TABLE_LOOKUP

```
/**
  This program defines the function that looks for the closest
  temperature value in the Thermistor table to the one read by the Arduino
*/

int table_lookup(int adc, int rows){

  //Variables
  int aux=0;
  int medium = rows/2;
  int i=0;
  int k=0;
```

```
//We check if the read value is closest to the bottom-half or top-half of the table
if (adc>thermtable[medium][0]){ //Bottom half
  k=0;
} else if (adc<thermtable[medium][0]){ //Top half
  k=medium;
}

//1º we try with the exact value
for (i=k; i < rows; i++) {
  if (adc==thermtable[i][0]){
    return thermtable[i][1]; //Exact value found!
  } else { aux++;}
}

i=k; //we make i= "0" or "medium" again

//2º we try with aprox values
int adc1=adc;
int low_adc=0;
int low_temp=0;
int upp_adc=0;
int upp_temp=0;
if (aux==rows){
  //We look for the upper value
  for (int j = 1; j < rows; j++) {
    adc1=adc+j;
    for (i=k; i < rows; i++) {
      if (adc1==thermtable[i][0]){
        upp_adc=thermtable[i][0]; //we store the lower adc value
        upp_temp=thermtable[i][1]; //we store the lower temp. value
        break;
      }
    }
    if (upp_adc!=0){ //If the upper value has been found: exist for(j)-loop
      break;
    }
  }
  //We look for the lower value (a partir del valor alto)
  for (int j = 1; j < rows; j++) {
    adc1=upp_adc-j;
    for (i = k; i < rows; i++) {
      if (adc1==thermtable[i][0] && thermtable[i][0]<upp_adc){
        low_adc=thermtable[i][0]; //we store the upper adc value
        low_temp=thermtable[i][1]; //we store the upper temp. value
        break;
      }
    }
    if (low_adc!=0){ //If the lower value has been found: exist for(j)-loop
      break;
    }
  }
  //We look for the closest value between the lower and upper value found
  if ((upp_adc-adc)<(adc-low_adc)){
    return upp_temp; //The upper value is closest
  } else {
    return low_temp; //The lower value is closest
  }
}
```

```
}
```

14.5.11. THERMTABLE_22

```
/**
 * Marlin inspired thermistor table
 */

// NTCLE305E4103SB
// 10kΩ thermistor at 25ºC

int thermtable[][2] = {
{ 16064, 150 },
{ 16000, 151 },
{ 15936, 152 },
{ 15872, 153 },
{ 15808, 154 },
{ 15744, 155 },
{ 15696, 156 },
{ 15632, 157 },
{ 15568, 158 },
{ 15504, 159 },
{ 15440, 160 },
{ 15376, 161 },
{ 15296, 162 },
{ 15232, 163 },
{ 15168, 164 },
{ 15120, 165 },
{ 15056, 166 },
{ 14992, 167 },
{ 14928, 168 },
{ 14864, 169 },
{ 14800, 170 },
{ 14736, 171 },
{ 14672, 172 },
{ 14608, 173 },
{ 14528, 174 },
{ 14464, 175 },
{ 14416, 176 },
{ 14352, 177 },
{ 14288, 178 },
{ 14224, 179 },
{ 14160, 180 },
{ 14096, 181 },
{ 14032, 182 },
{ 13968, 183 },
{ 13904, 184 },
{ 13840, 185 },
{ 13776, 186 },
{ 13712, 187 },
{ 13648, 188 },
{ 13584, 189 },
{ 13520, 190 },
{ 13456, 191 },
{ 13376, 192 },
{ 13328, 193 },
{ 13264, 194 },
{ 13200, 195 },
```

```
{ 13136, 196 },
{ 13072, 197 },
{ 13008, 198 },
{ 12944, 199 },
{ 12880, 200 },
{ 12800, 201 },
{ 12736, 202 },
{ 12672, 203 },
{ 12608, 204 },
{ 12544, 205 },
{ 12480, 206 },
{ 12416, 207 },
{ 12352, 208 },
{ 12288, 209 },
{ 12224, 210 },
{ 12160, 211 },
{ 12096, 212 },
{ 12032, 213 },
{ 11968, 214 },
{ 11904, 215 },
{ 11840, 216 },
{ 11776, 217 },
{ 11712, 218 },
{ 11648, 219 },
{ 11584, 220 },
{ 11520, 221 },
{ 11456, 222 },
{ 11392, 223 },
{ 11328, 224 },
{ 11264, 225 },
{ 11184, 226 },
{ 11136, 227 },
{ 11072, 228 },
{ 11008, 229 },
{ 10944, 230 },
{ 10880, 231 },
{ 10816, 232 },
{ 10752, 233 },
{ 10688, 234 },
{ 10624, 235 },
{ 10560, 236 },
{ 10496, 237 },
{ 10432, 238 },
{ 10368, 239 },
{ 10304, 240 },
{ 10240, 241 },
{ 10176, 242 },
{ 10128, 243 },
{ 10064, 244 },
{ 9984, 245 },
{ 9936, 246 },
{ 9872, 247 },
{ 9792, 248 },
{ 9728, 249 },
{ 9680, 250 },
{ 9616, 251 },
{ 9536, 252 },
{ 9488, 253 },
{ 9408, 254 },
```

```
{ 9344, 255 },
{ 9280, 256 },
{ 9216, 257 },
{ 9168, 258 },
{ 9104, 259 },
{ 9040, 260 },
{ 8976, 261 },
{ 8912, 262 },
{ 8848, 263 },
{ 8784, 264 },
{ 8720, 265 },
{ 8672, 266 },
{ 8608, 267 },
{ 8544, 268 },
{ 8480, 269 },
{ 8416, 270 },
{ 8352, 271 },
{ 8288, 272 },
{ 8224, 273 },
{ 8160, 274 },
{ 8112, 275 },
{ 8032, 276 },
{ 7968, 277 },
{ 7920, 278 },
{ 7856, 279 },
{ 7792, 280 },
{ 7744, 281 },
{ 7664, 282 },
{ 7600, 283 },
{ 7536, 284 },
{ 7472, 285 },
{ 7424, 286 },
{ 7376, 287 },
{ 7296, 288 },
{ 7248, 289 },
{ 7184, 290 },
{ 7120, 291 },
{ 7056, 292 },
{ 6992, 293 },
{ 6928, 294 },
{ 6880, 295 },
{ 6816, 296 },
{ 6752, 297 },
{ 6688, 298 },
{ 6640, 299 },
{ 6576, 300 },
{ 6512, 301 },
{ 6464, 302 },
{ 6400, 303 },
{ 6320, 304 },
{ 6256, 305 },
{ 6208, 306 },
{ 6144, 307 },
{ 6080, 308 },
{ 6016, 309 },
{ 5952, 310 },
{ 5888, 311 },
{ 5840, 312 },
{ 5776, 313 },
```

```
{ 5712, 314 },
{ 5648, 315 },
{ 5600, 316 },
{ 5536, 317 },
{ 5488, 318 },
{ 5424, 319 },
{ 5360, 320 },
{ 5296, 321 },
{ 5232, 322 },
{ 5168, 323 },
{ 5120, 324 },
{ 5056, 325 },
{ 5008, 326 },
{ 4944, 327 },
{ 4880, 328 },
{ 4800, 329 },
{ 4752, 330 },
{ 4688, 331 },
{ 4640, 332 },
{ 4592, 333 },
{ 4528, 334 },
{ 4464, 335 },
{ 4400, 336 },
{ 4352, 337 },
{ 4288, 338 },
{ 4224, 339 },
{ 4160, 340 },
{ 4096, 341 },
{ 4048, 342 },
{ 4000, 343 },
{ 3936, 344 },
{ 3872, 345 },
{ 3824, 346 },
{ 3760, 347 },
{ 3712, 348 },
{ 3648, 349 },
{ 3600, 350 },
{ 3536, 351 },
{ 3472, 352 },
{ 3408, 353 },
{ 3344, 354 },
{ 3296, 355 },
{ 3232, 356 },
{ 3184, 357 },
{ 3136, 358 },
{ 3088, 359 },
{ 3024, 360 },
{ 2960, 361 },
{ 2912, 362 },
{ 2848, 363 },
{ 2800, 364 },
{ 2736, 365 },
{ 2688, 366 },
{ 2624, 367 },
{ 2576, 368 },
{ 2512, 369 },
{ 2448, 370 },
{ 2400, 371 },
{ 2336, 372 },
```

```
{ 2288, 373 },
{ 2240, 374 },
{ 2176, 375 },
{ 2128, 376 },
{ 2080, 377 },
{ 2016, 378 },
{ 1968, 379 },
{ 1888, 380 },
{ 1840, 381 },
{ 1792, 382 },
{ 1776, 383 },
{ 1728, 384 },
{ 1664, 385 },
{ 1584, 386 },
{ 1520, 387 },
{ 1472, 388 },
{ 1424, 389 },
{ 1376, 390 },
{ 1344, 391 },
{ 1296, 392 },
{ 1232, 393 },
{ 1168, 394 },
{ 1104, 395 },
{ 1040, 396 },
{ 992, 397 },
{ 944, 398 },
{ 896, 399 },
{ 832, 400 },
};
```

14.5.12. THERMTABLES

```
/**
 This program defines the correct table for the selected thermistor
 This allows for different thermistors to be implemented easily
*/

#define ANY_THERMISTOR_IS(n) (ThermSensor == n)

// Include here your thermistor-table

 #if ANY_THERMISTOR_IS(22) // 10k thermistor 0,1ªC sensibility
  #include "thermtable_22.h"
 #endif

 #if ANY_THERMISTOR_IS(23) // 10k thermistor 0,5ºC sensibility
  #include "thermtable_23.h"
 #endif
```

14.5.13. TIMER

```
/**
 * This library includes the function that deals with the timer set for the movement of the
motors.
 *
 * The following function id included here:
 *    void timer_set_frequency(long desired_freq_hz)
```

```
 *   ISR(TIMER1_COMPA_vect)
 *
 */

/**
 * Set the clock 1 timer frequency.
 * @input desired_freq_hz the desired frequency
 */
void timer_set_frequency(long desired_freq_hz) {
  // Source: http://letsmakerobots.com/node/28278
  // Different clock sources can be selected for each timer independently.
  // To calculate the timer frequency (for example 2Hz using timer1) you will need:

  if(desired_freq_hz>20000) {
    step_loops=4;
    desired_freq_hz>>=2;
  } else if(desired_freq_hz>10000) {
    step_loops=2;
    desired_freq_hz>>=1;
  } else {
    step_loops=1;
  }

  //  CPU frequency 16Mhz for Arduino
  //  maximum timer counter value (256 for 8bit, 65536 for 16bit timer)
  int prescaler_index=-1;
  int prescalers[] = {1,8,64,256,1024};
  long counter_value;
  do {
    ++prescaler_index;
    //  Divide CPU frequency through the choosen prescaler (16000000 / 256 = 62500)
    counter_value = CLOCK_FREQ / prescalers[prescaler_index];
    //  Divide result through the desired frequency (62500 / 2Hz = 31250)
    counter_value /= desired_freq_hz;
    //  Verify counter_value < maximum timer. if fail, choose bigger prescaler.
  } while(counter_value > MAX_COUNTER && prescaler_index<4);

  if( prescaler_index>=5 ) {
    Serial.println(F("Timer could not be set: Desired frequency out of bounds."));
    return;
  }

  // disable global interrupts
  noInterrupts();

  // set entire TCCR1A register to 0
  TCCR1A = 0;
  // set entire TCCR1B register to 0
  TCCR1B = 0;
  // set the overflow clock to 0
  TCNT1  = 0;
  // set compare match register to desired timer count
  OCR1A = counter_value;
  // turn on CTC mode
  TCCR1B |= (1 << WGM12);
  // Set CS10, CS11, and CS12 bits for prescaler
  TCCR1B |= ( (( prescaler_index&0x1 )   ) << CS10);
  TCCR1B |= ( (( prescaler_index&0x2 )>>1) << CS11);
  TCCR1B |= ( (( prescaler_index&0x4 )>>2) << CS12);
```

```
  // enable timer compare interrupt
  TIMSK1 |= (1 << OCIE1A);

  interrupts();  // enable global interrupts
}


/**
 * Process all line segments in the ring buffer.  Uses bresenham's line algorithm to move all
motors.
 */
ISR(TIMER1_COMPA_vect) {
  // segment buffer empty? do nothing
  if( current_segment == last_segment ) return;

  // Is this segment done?
  if( line_segments[current_segment].steps_left <= 0 ) {
    // Move on to next segment without wasting an interrupt tick.
    current_segment = get_next_segment(current_segment);
    if( current_segment == last_segment ) return;
  }

  int j;
  Segment &seg = line_segments[current_segment];
  // is this a fresh new segment?
  if( seg.steps == seg.steps_left ) {
    // set the direction pins
    for(j=0;j<NUM_AXIES;++j) {
      digitalWrite( motors[j].dir_pin, line_segments[current_segment].a[j].dir );
      seg.a[j].over = seg.steps/2;
    }
    // set frequency to segment feed rate
    timer_set_frequency(seg.feed_rate);
  }

  for(int s=0;s<step_loops;++s) {
    // make a step
    --seg.steps_left;

    // move each axis
    // TODO unroll this loop and dereference earlier to make this faster
    // will raise the top speed of the machine.
    for(j=0;j<NUM_AXIES;++j) {
      Axis &a = seg.a[j];

      a.over += a.absdelta;
      if(a.over >= seg.steps) {
        digitalWrite(motors[j].step_pin,LOW);
        a.over -= seg.steps;
        digitalWrite(motors[j].step_pin,HIGH);
      }
    }
  }
}
```